



Heap Abstraction via **Early-Confluent** Object Merging for Pointer Analysis

Jinpeng Wang*, Yufei Liang*, Zhongsheng Zhan

Tian Tan† and Yue Li†



*Equal contribution.

†Corresponding authors.

Pointer Analysis

- Answers

Which **objects** a variable may point to.

- Importance

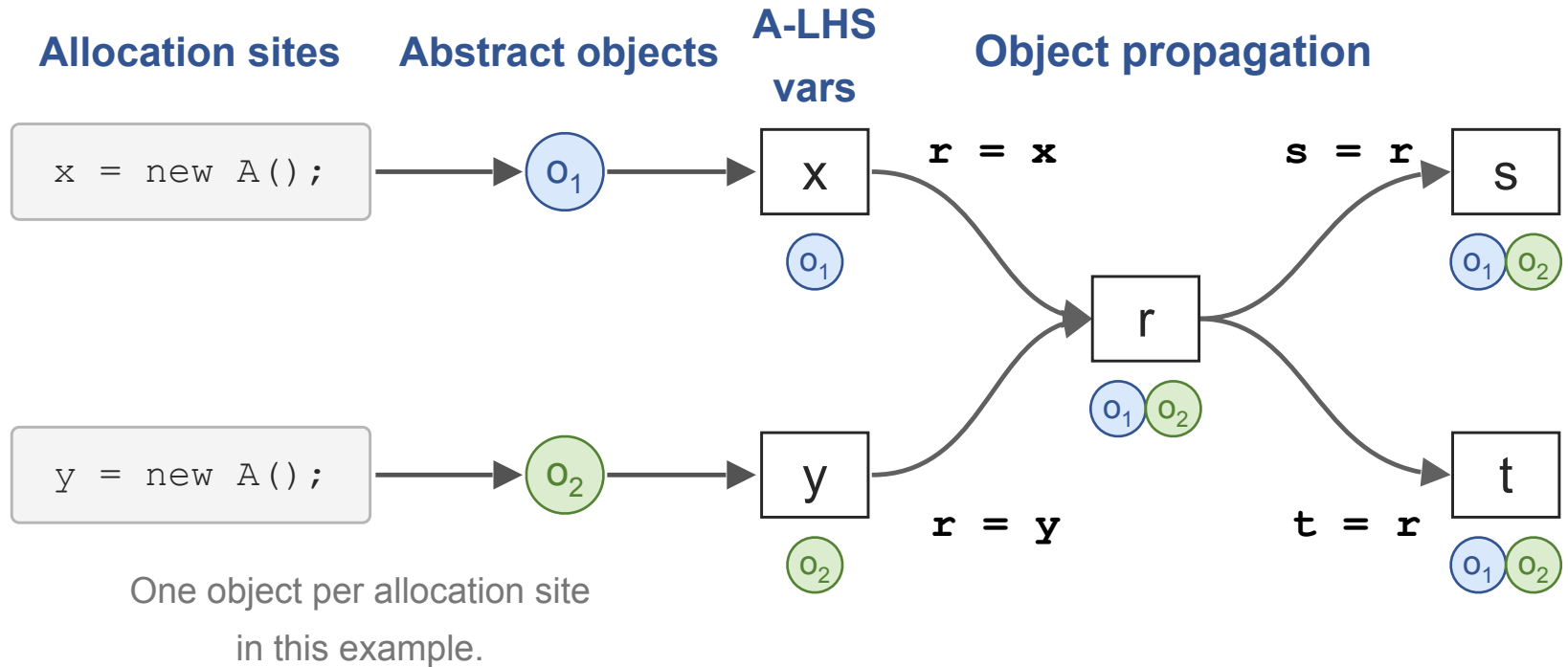
A foundation for many static analyses

e.g., call graphs, aliases, side effects, etc.

Supports many useful analysis clients

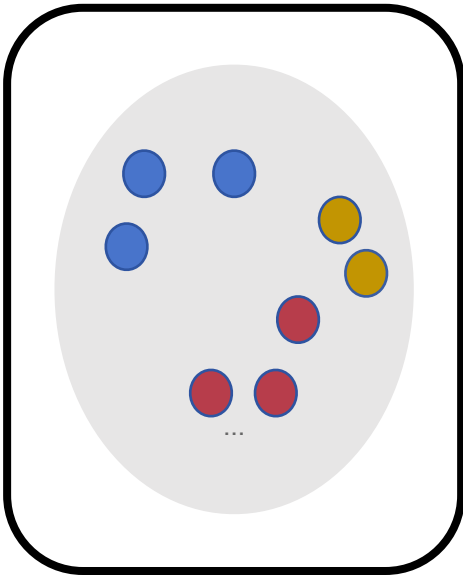
e.g., bug detection, compiler optimization, security analysis, etc.

How Pointer Analysis Works



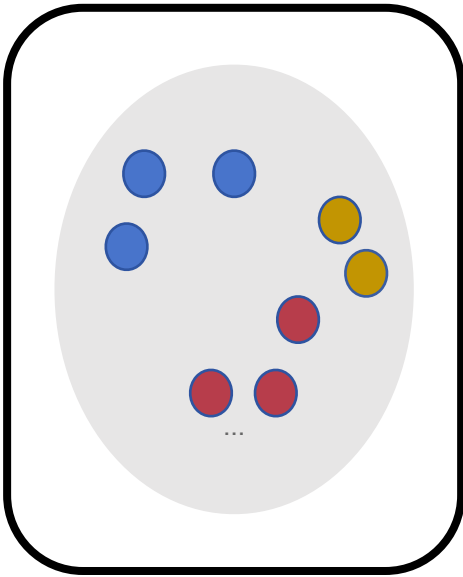
Graph reachability: $O(|O| \times (|V| + |E|))$

How Heap Abstraction Works

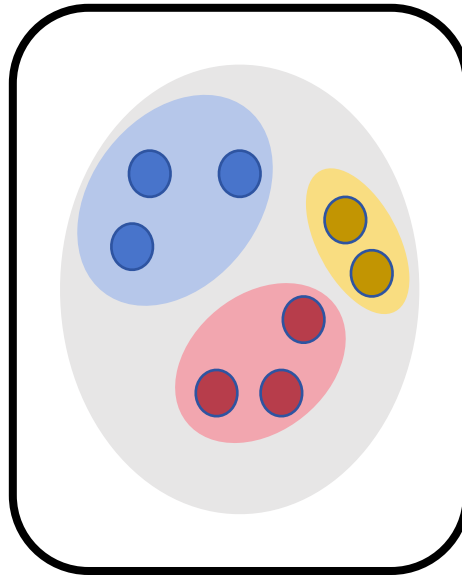


- Dynamic Heap
8 concrete objects

How Heap Abstraction Works

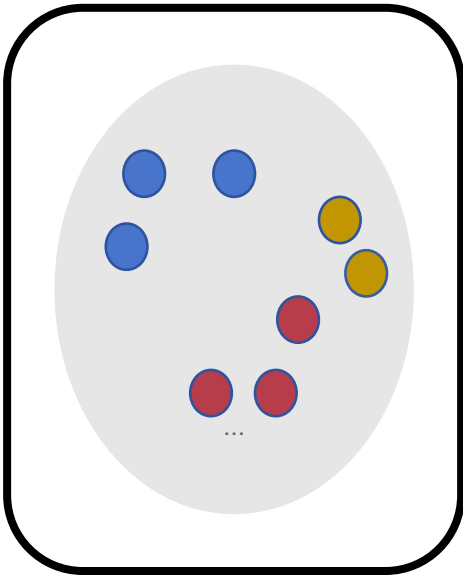


- Dynamic Heap
8 concrete objects

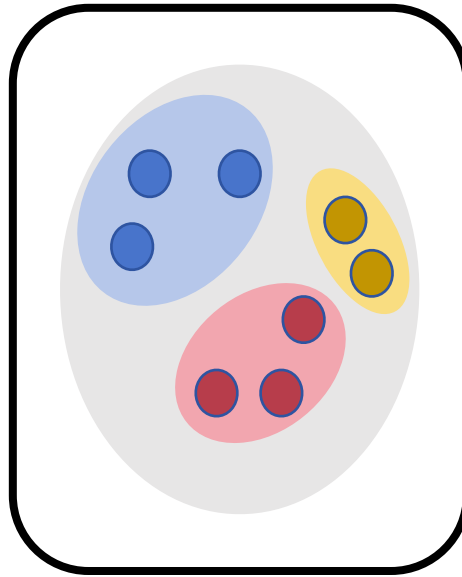


- Abstract Heap
3 abstract objects

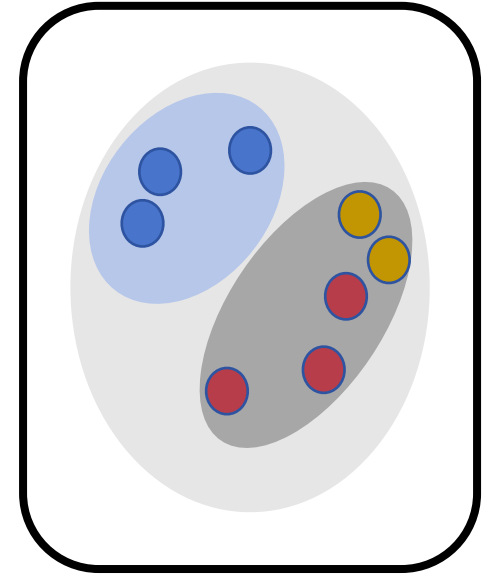
How Heap Abstraction Works



- Dynamic Heap
8 concrete objects



- Abstract Heap
3 abstract objects



- Merged Heap
2 abstract objects

Which Objects Should We Merge?

- Fewer objects → less propagation

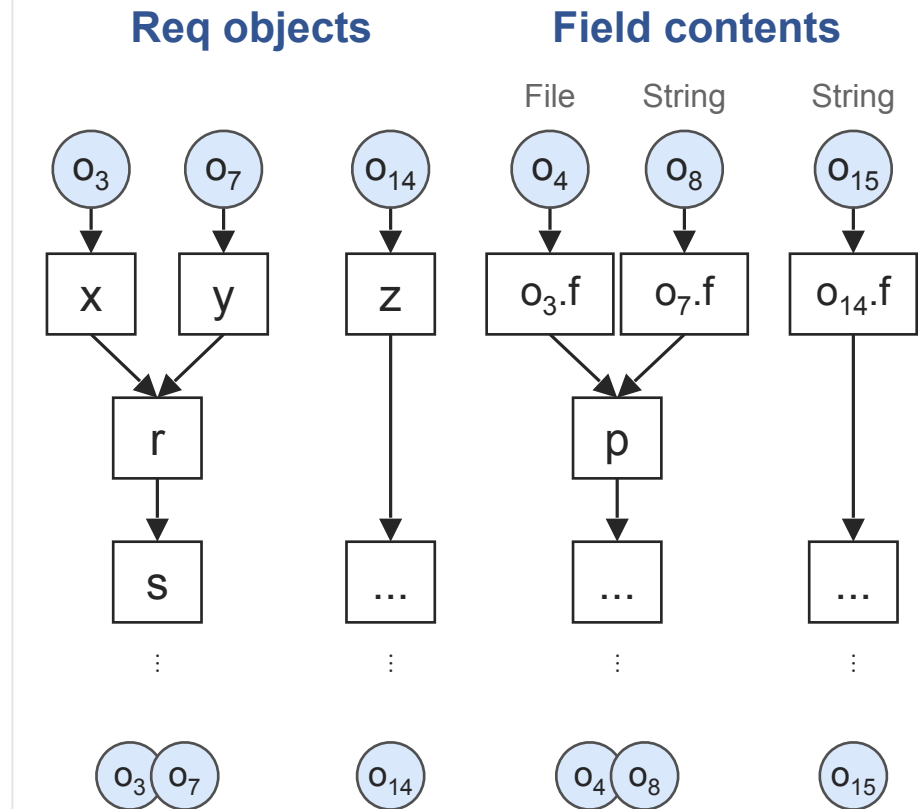
Yet also introduces imprecision.

- Client-specific solution is far from satisfactory

Some clients impose higher demands on points-to precision than others.

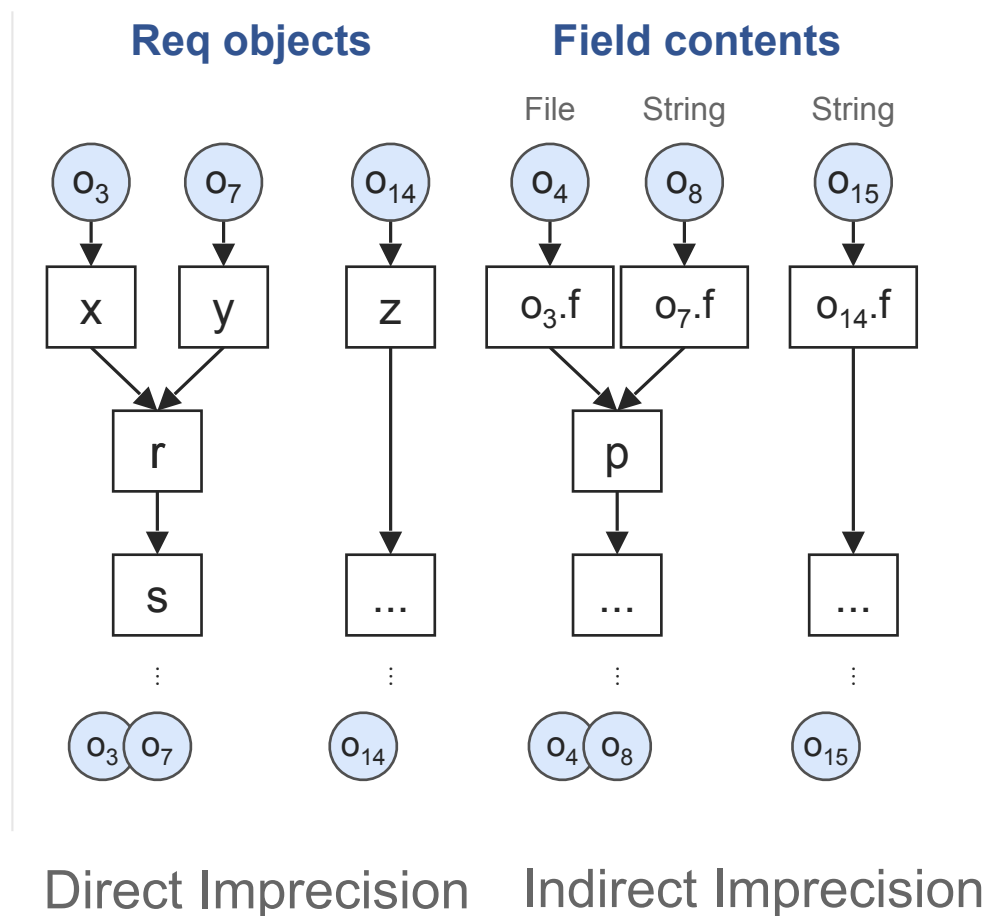
Example: Without Merging

```
1 Req build() {  
2   if (...) {  
3     x = new Req();      // o3  
4     x.f = new File();   // o4  
5     return x;  
6   } else {  
7     y = new Req();      // o7  
8     y.f = new String(); // o8  
9     return y;  
10  }  
11 }  
  
12 r = build();  
13 s = r; p = s.f;  
14 z = new Req();          // o14  
15 z.f = new String();     // o15
```



Mahjong: Type-Consistent Merging

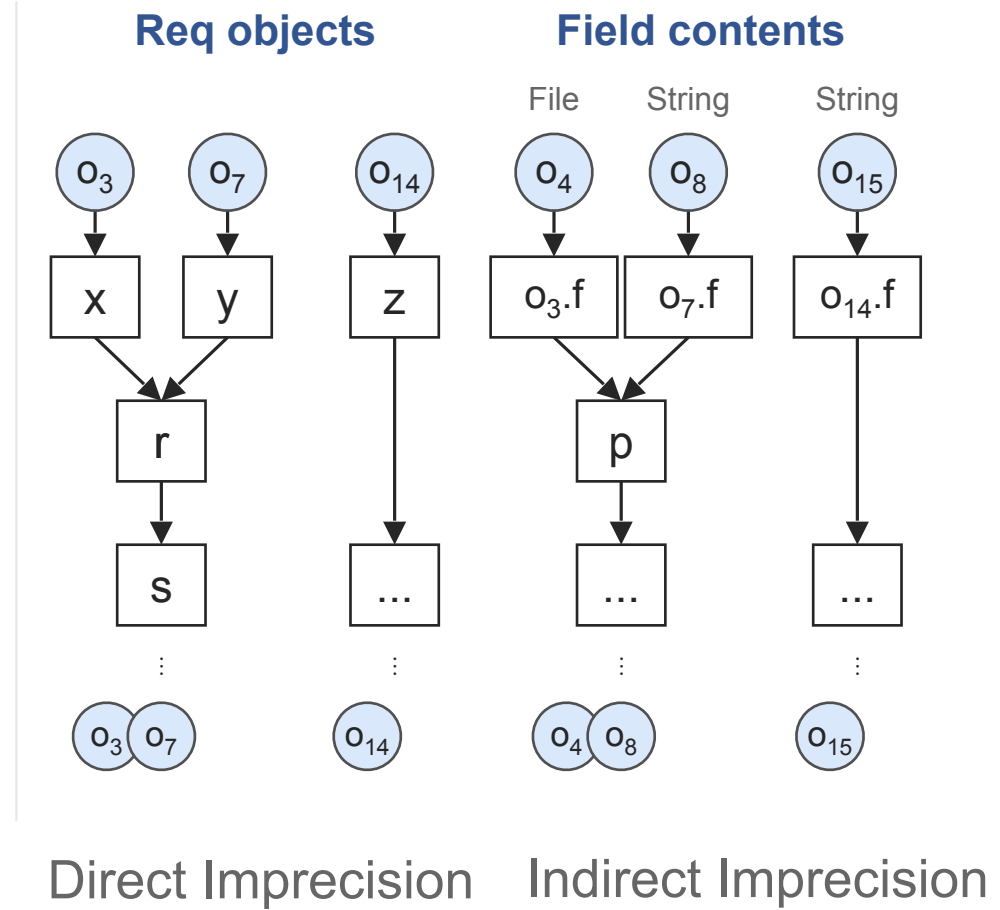
What happens if we merge o_7 and o_{14} ?



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

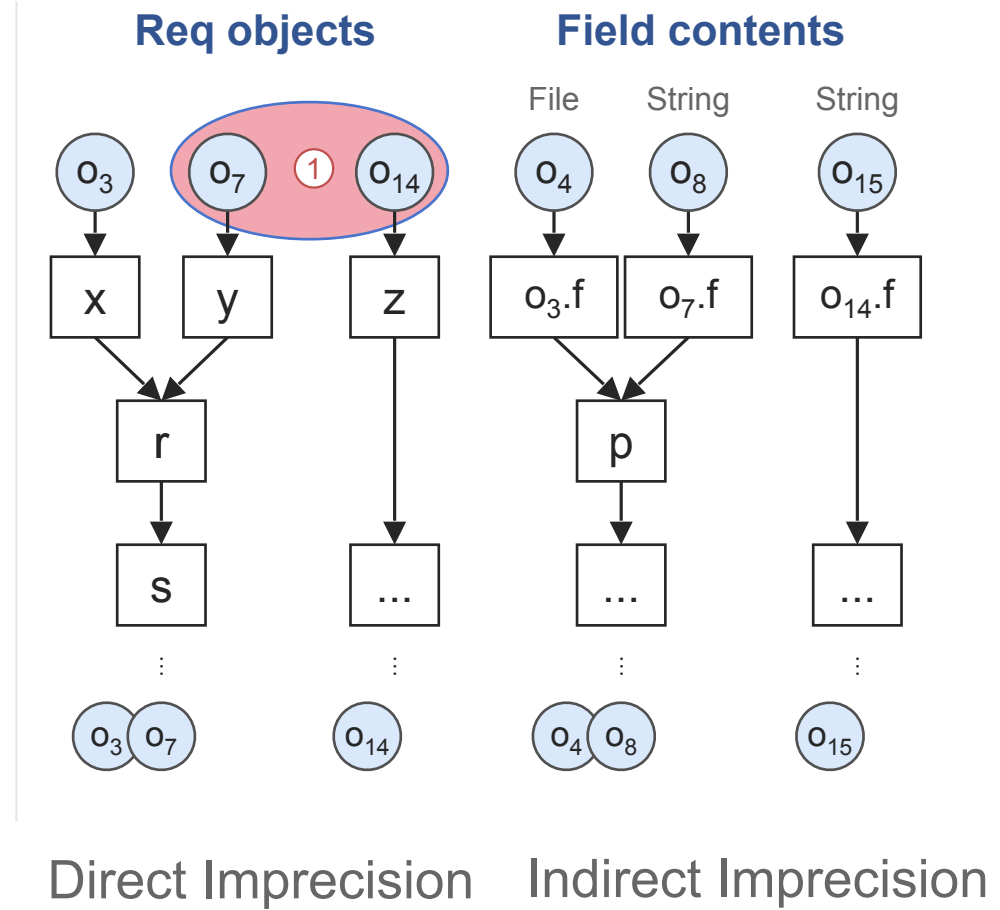
① y points to o_{14} , z points to o_7



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

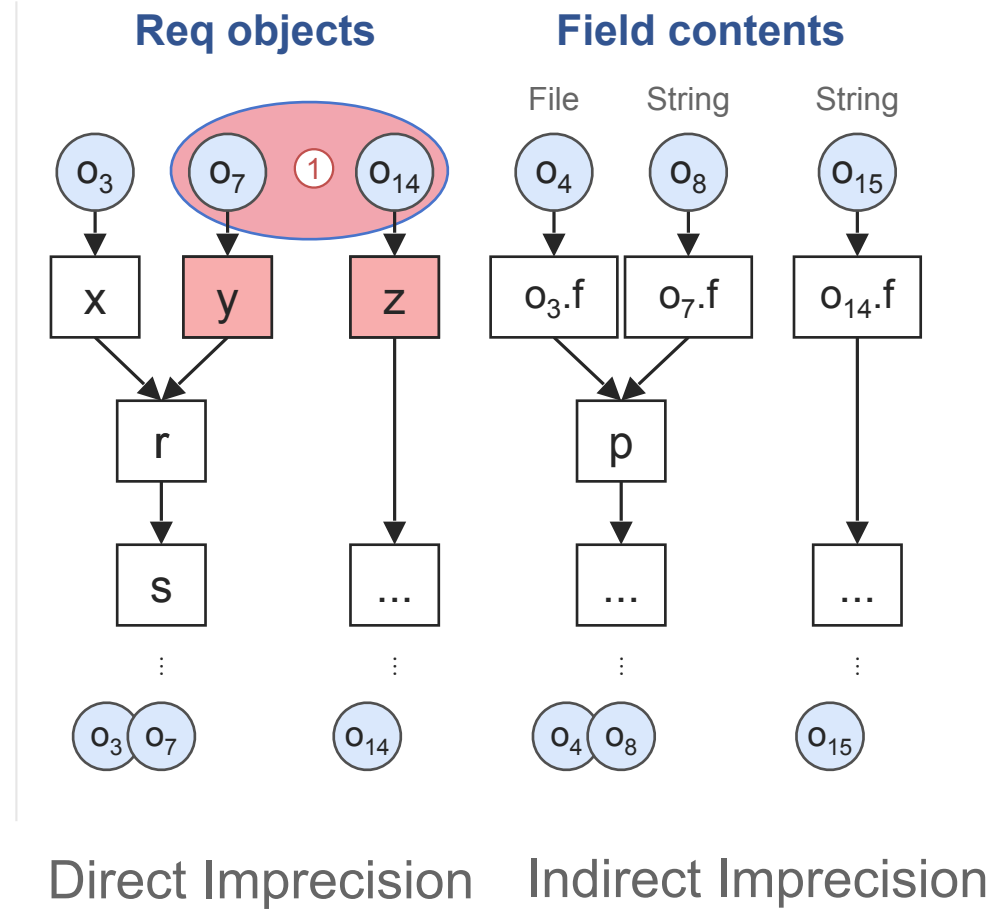
① y points to o_{14} , z points to o_7



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

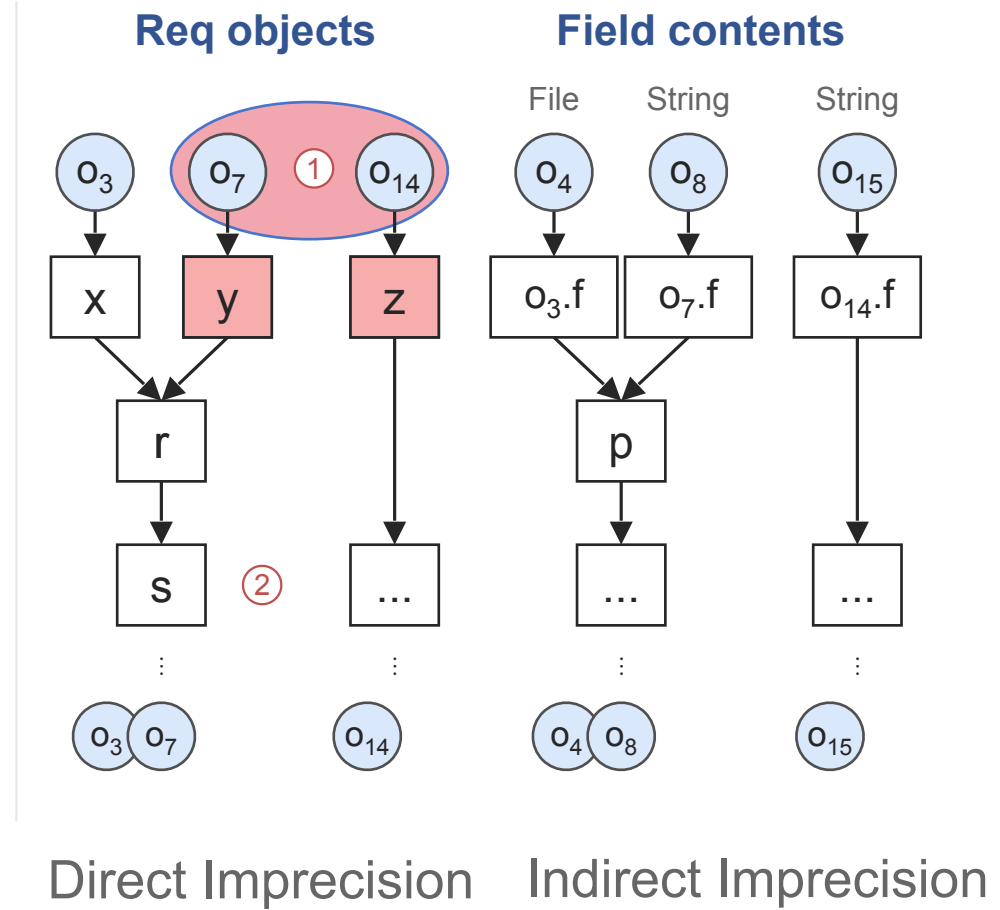
① y points to o_{14} , z points to o_7



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

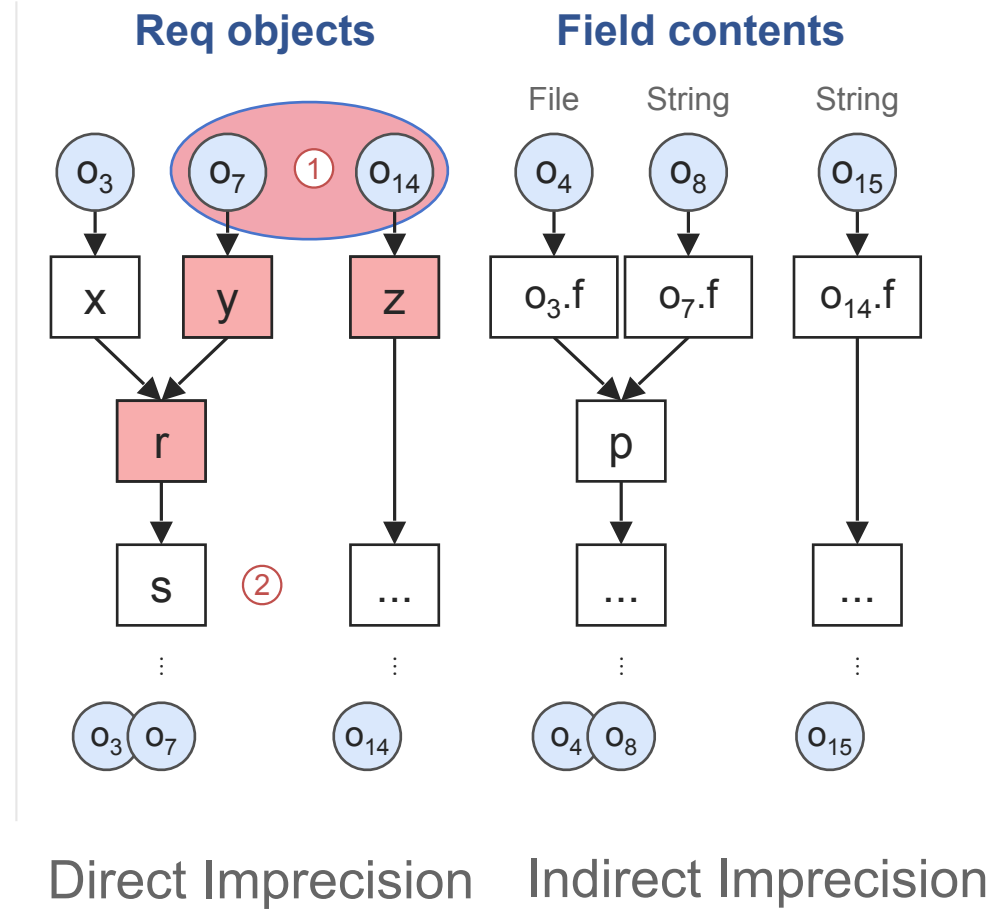
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

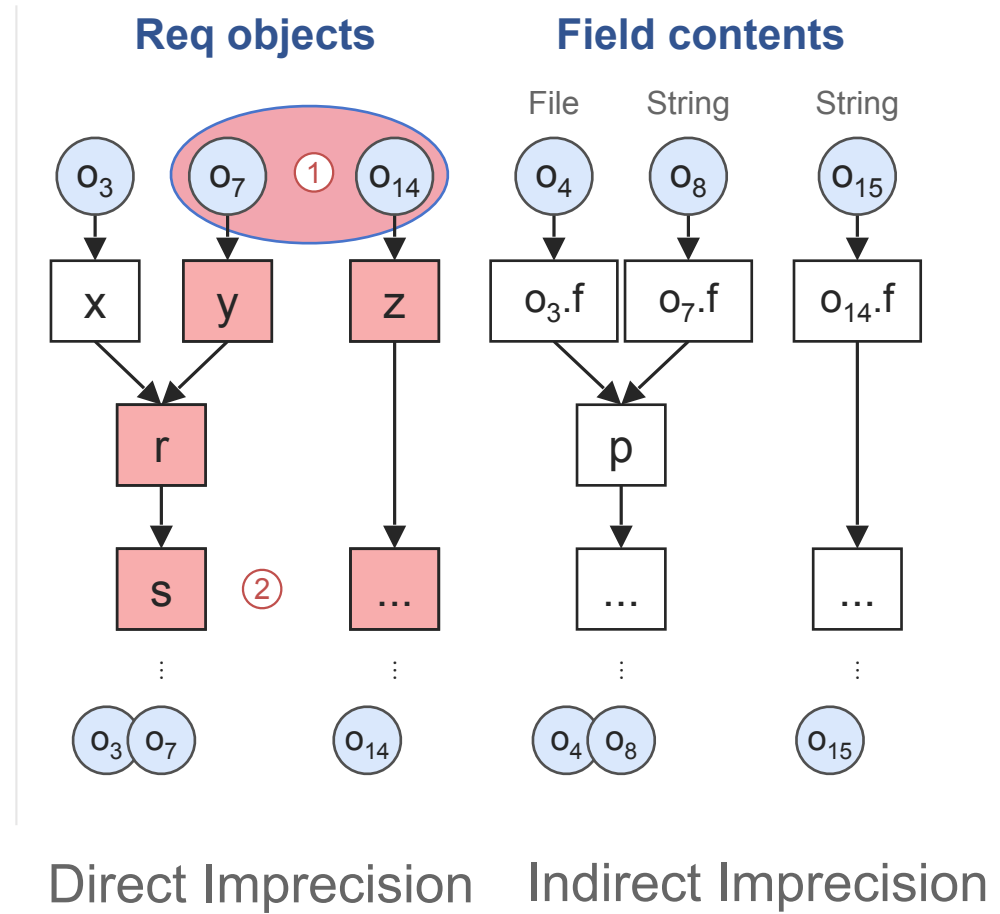
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

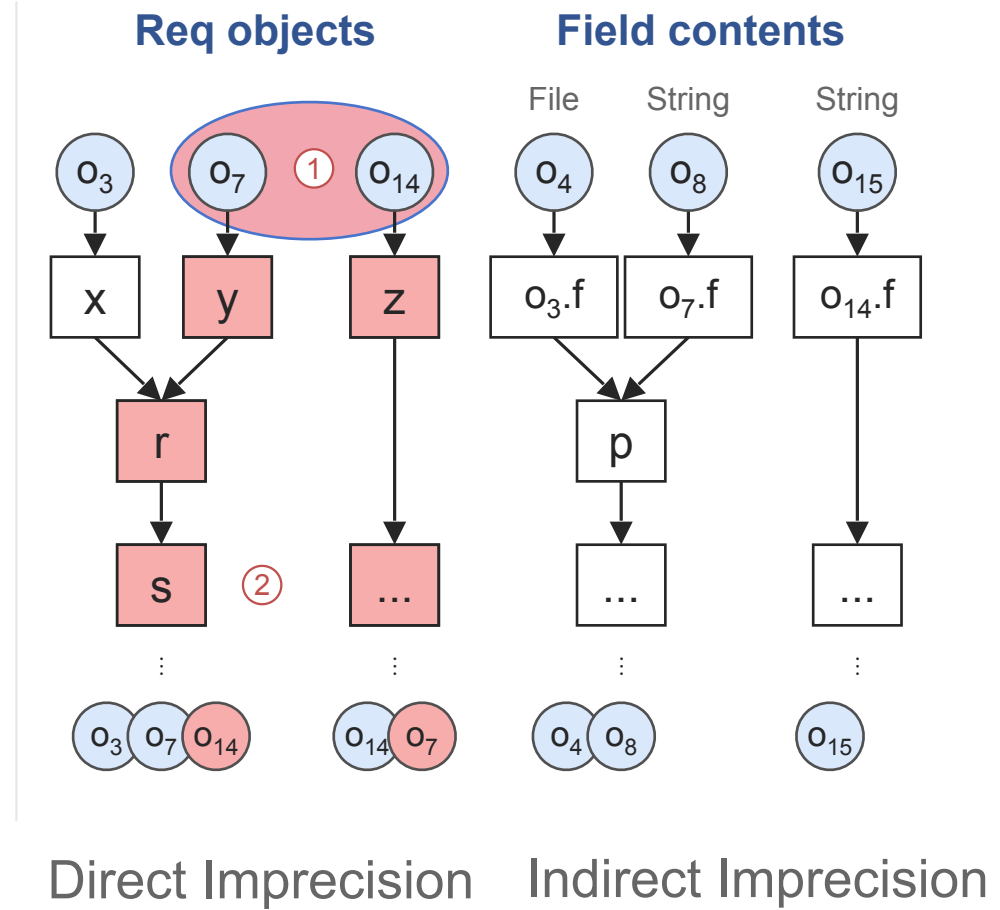
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

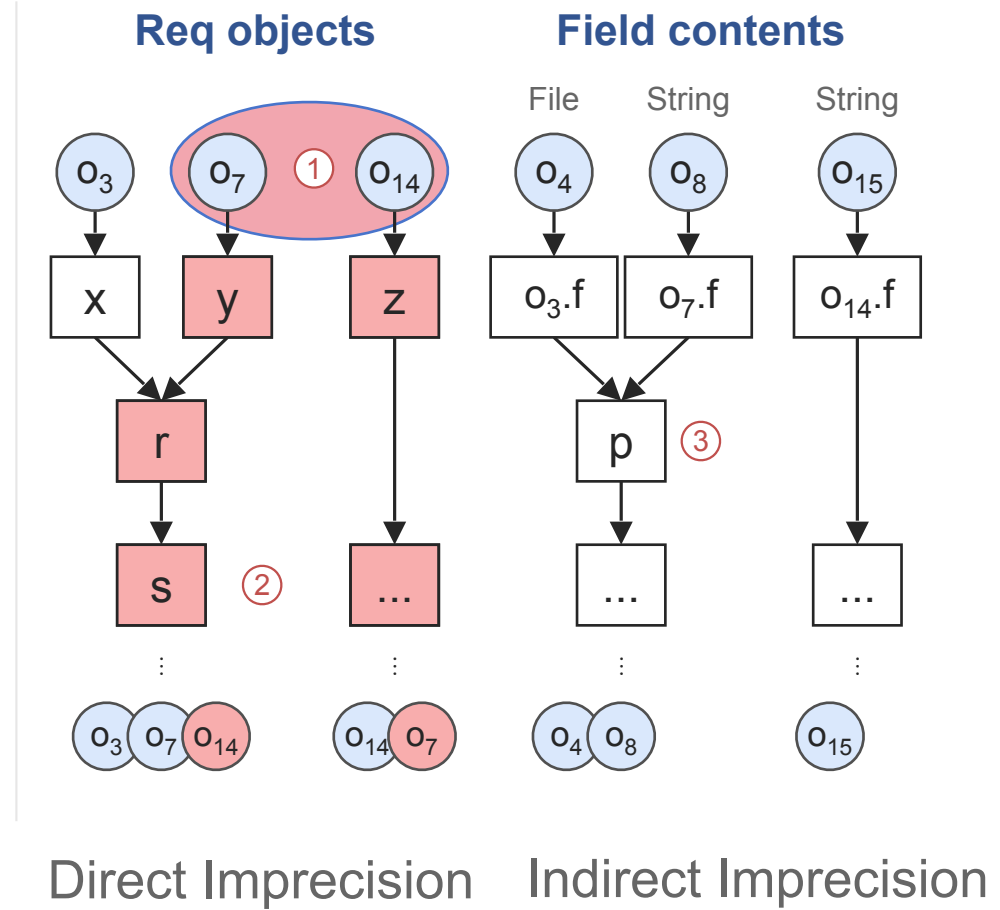
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

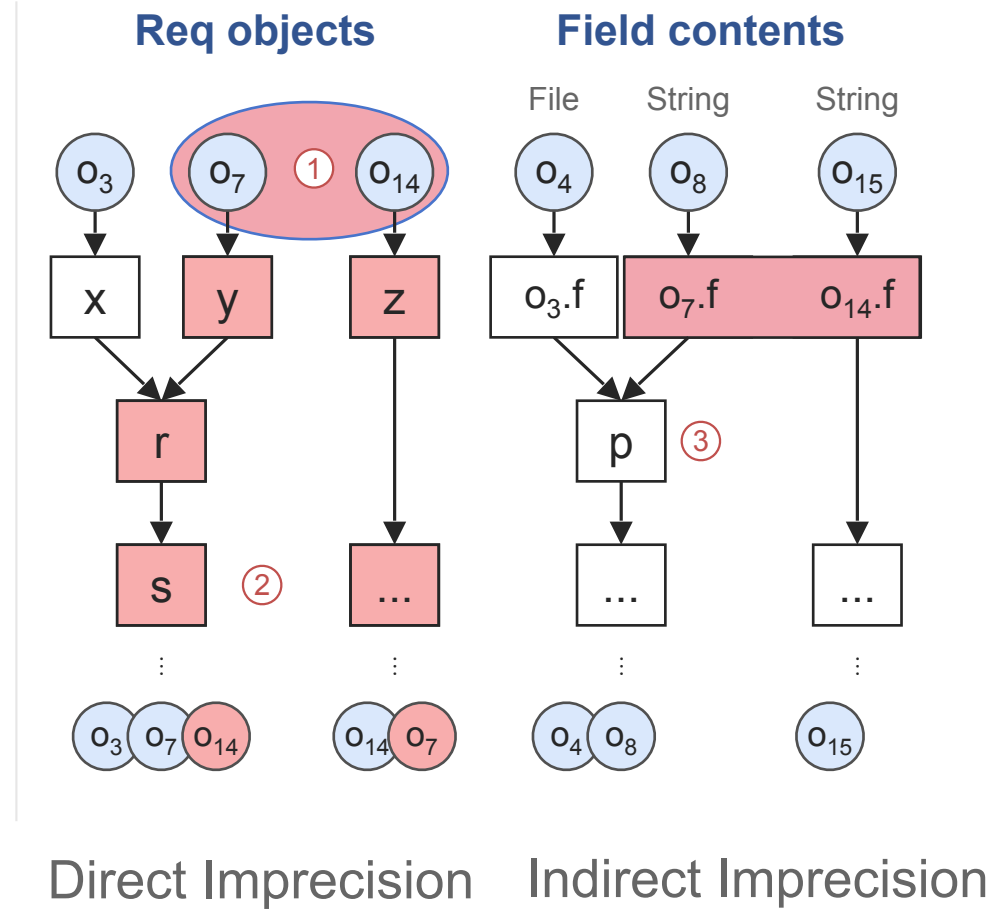
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).
- ③ $p = s.f$ also points to o_{15}



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

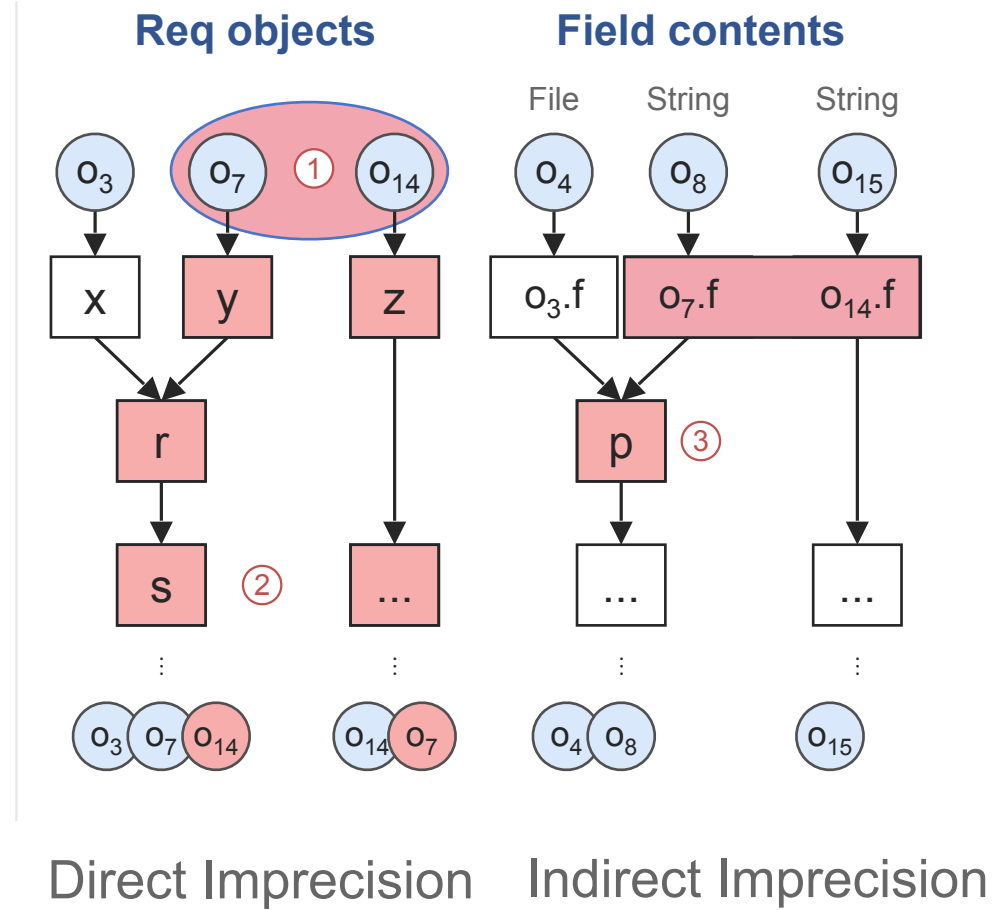
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).
- ③ $p = s.f$ also points to o_{15}



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

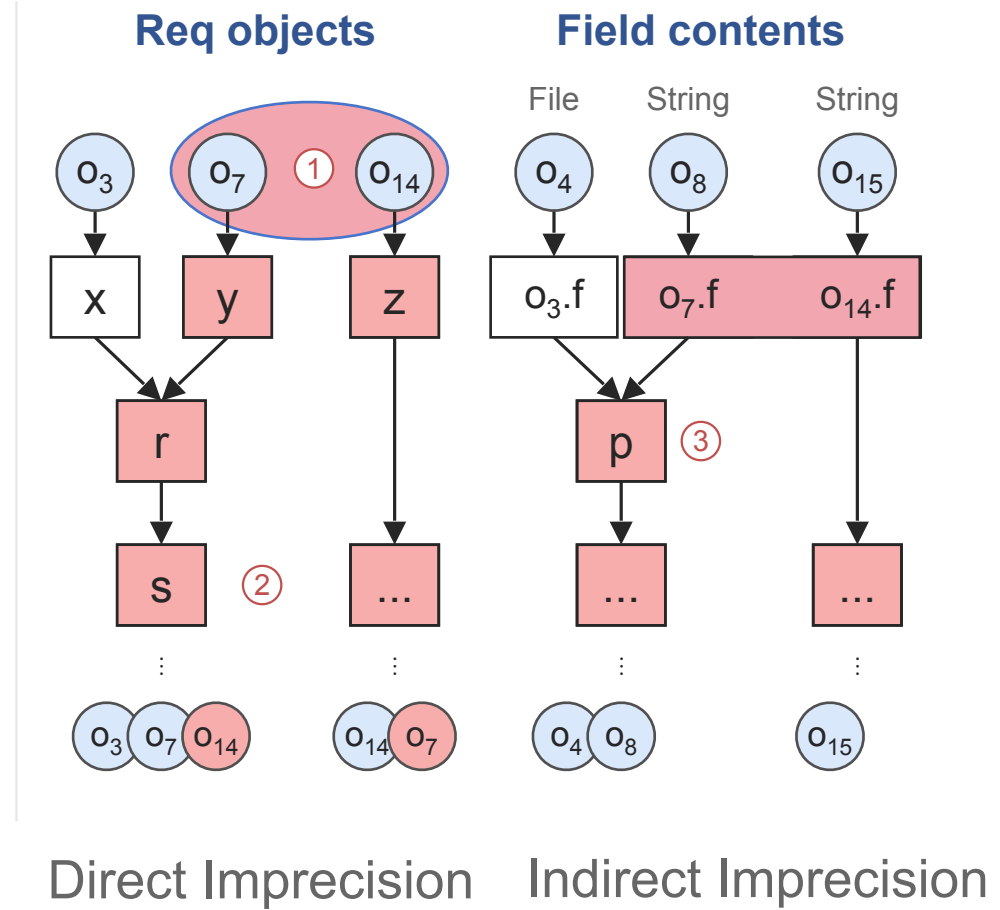
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).
- ③ $p = s.f$ also points to o_{15}



Mahjong: Type-Consistent Merging

What happens if we merge o_7 and o_{14} ?

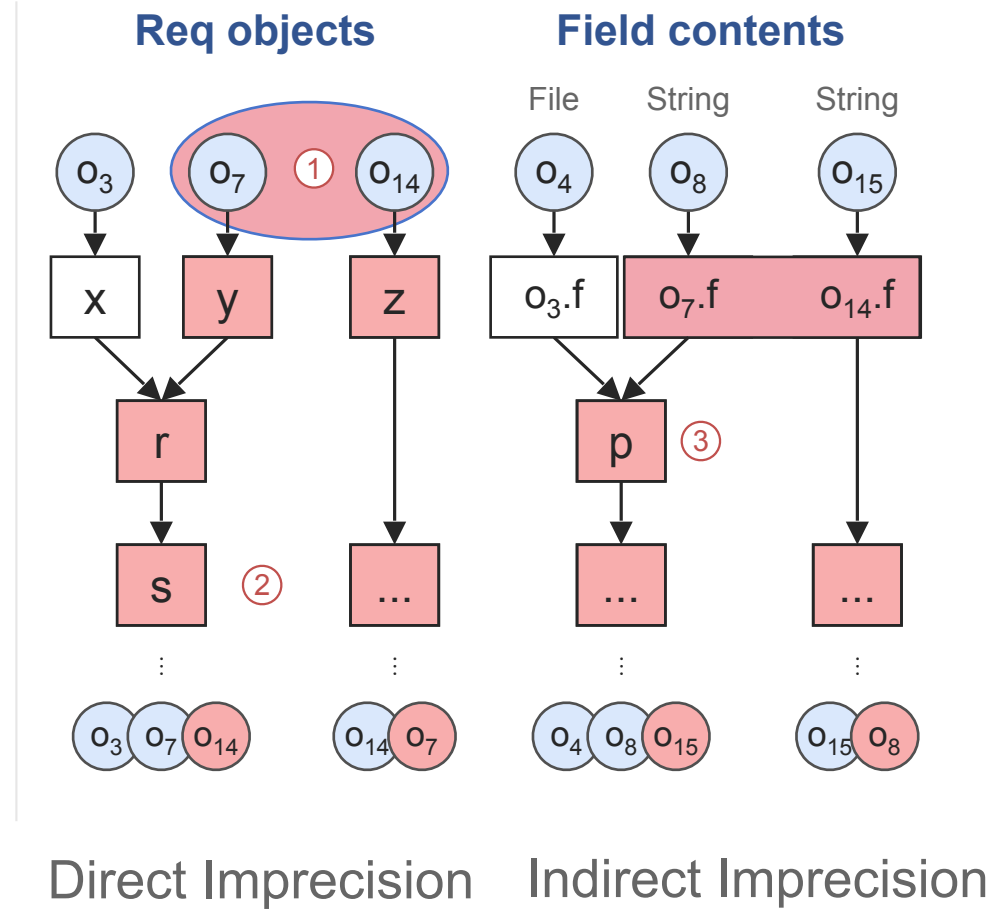
- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).
- ③ $p = s.f$ also points to o_{15}



Mahjong: Type-Consistent Merging

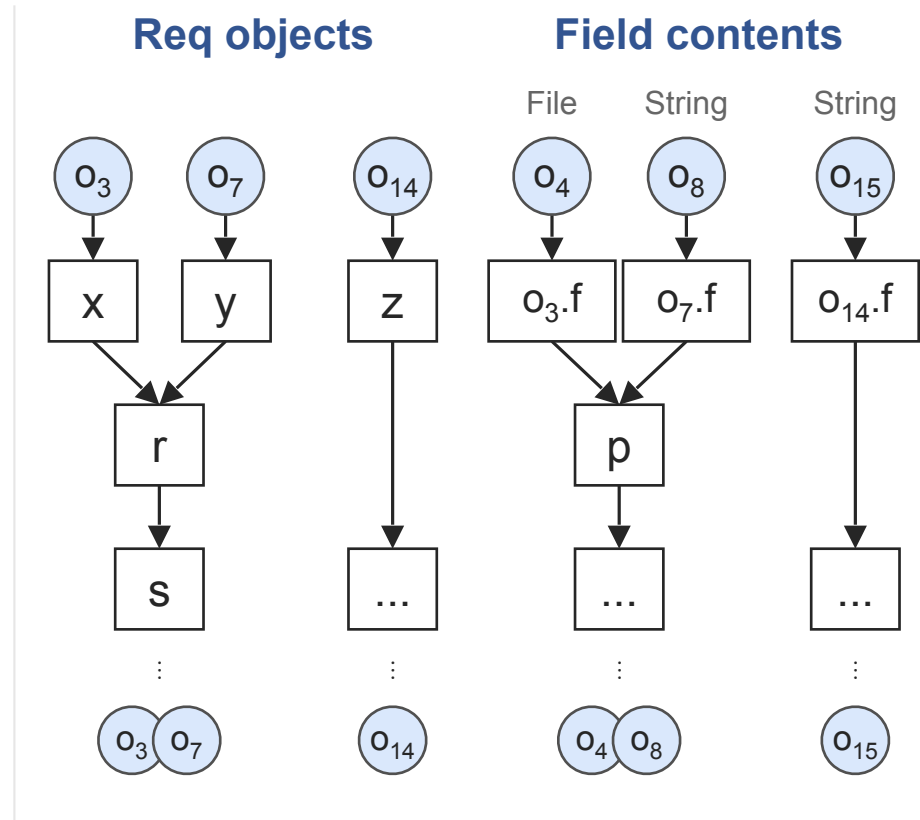
What happens if we merge o_7 and o_{14} ?

- ① y points to o_{14} , z points to o_7
- ② $o_{14}(o_7)$ flows to r and s (z 's successors).
- ③ $p = s.f$ also points to o_{15}



Valve: Early-Confluent Merging

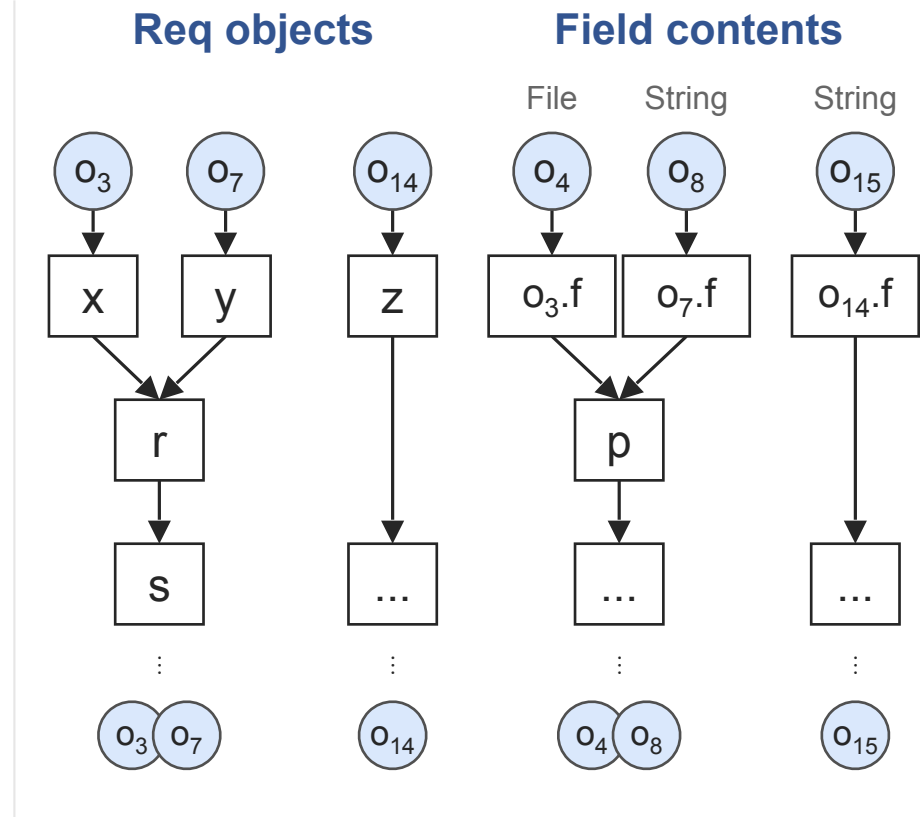
What happens if we merge o_3 and o_7 ?



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

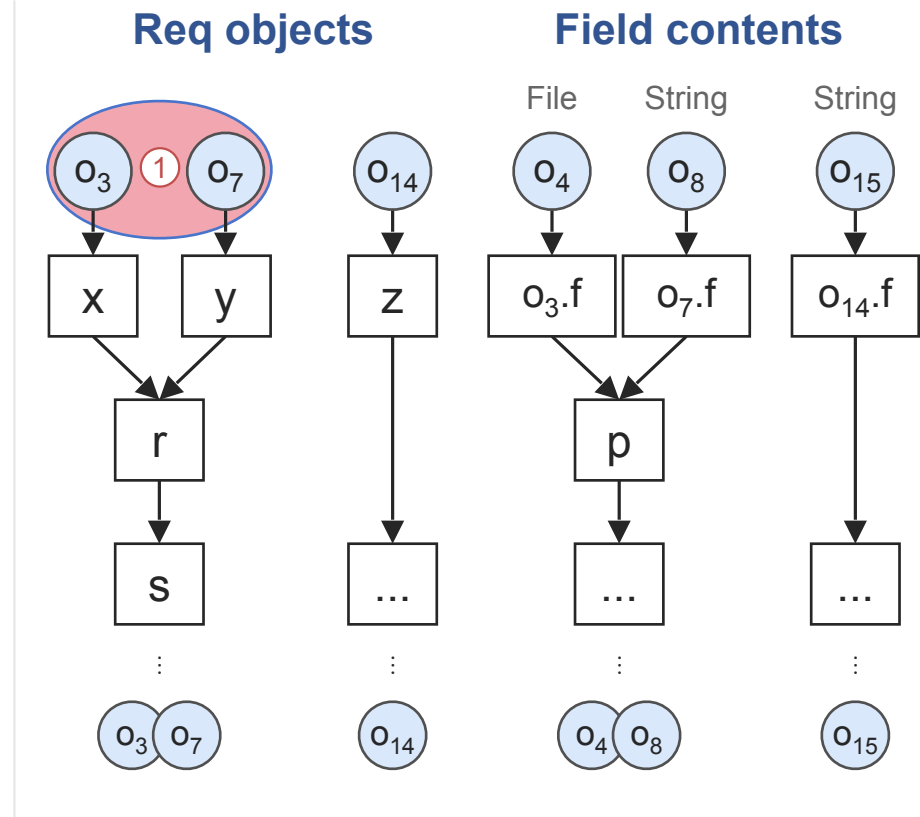
① x points to o_7 , y points to o_3



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

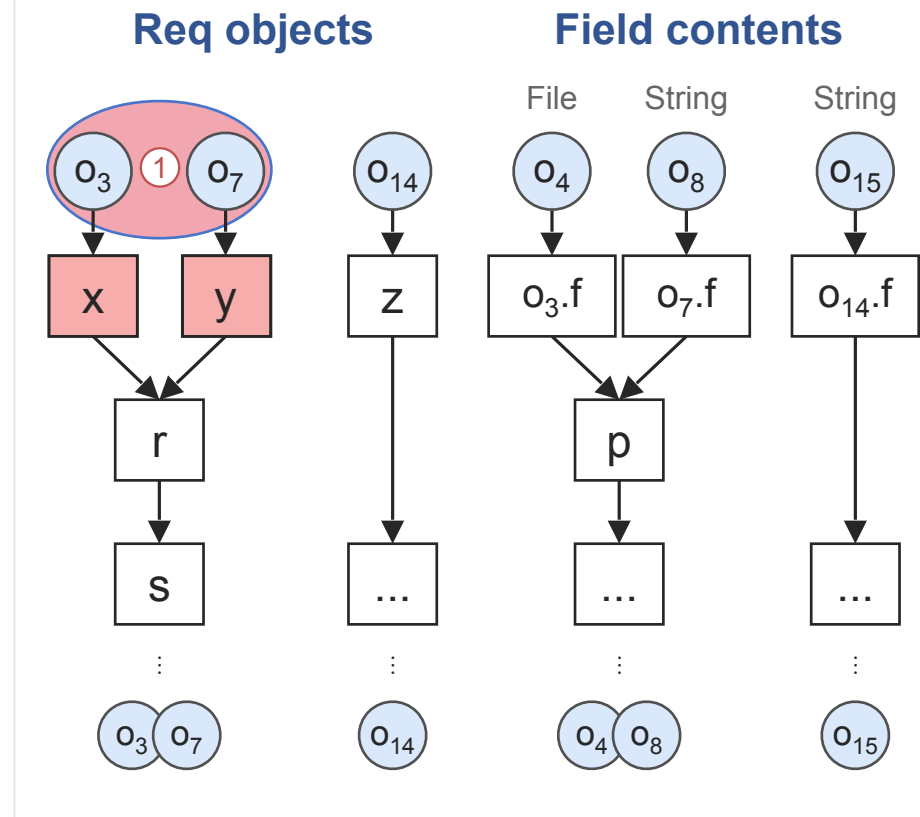
① x points to o_7 , y points to o_3



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

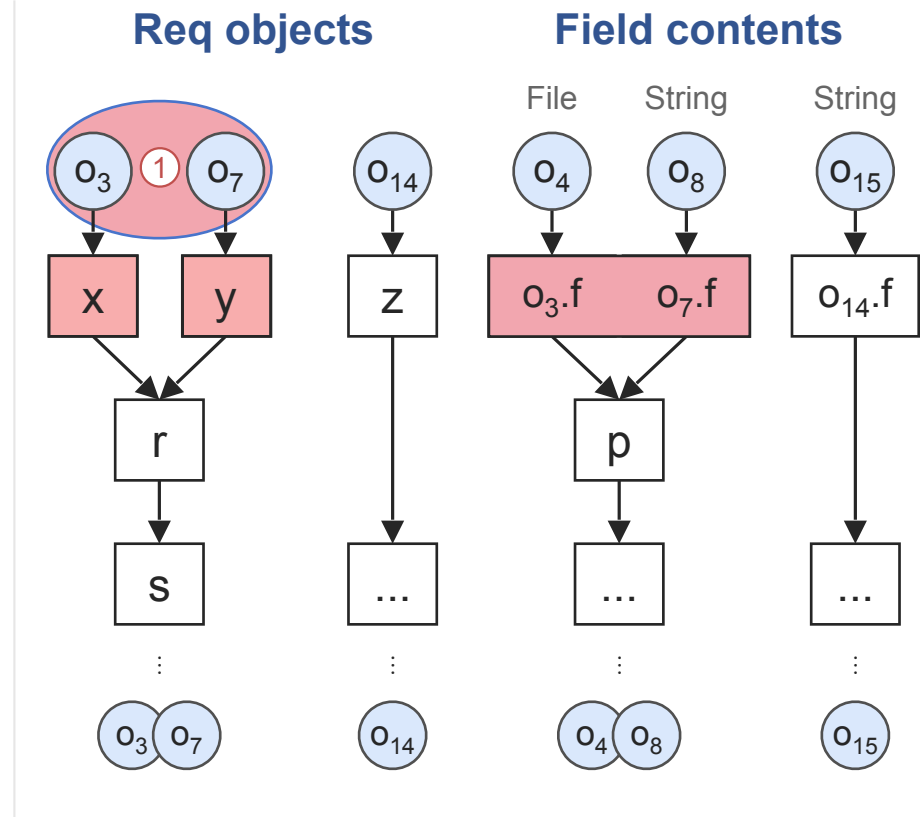
① x points to o_7 , y points to o_3



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

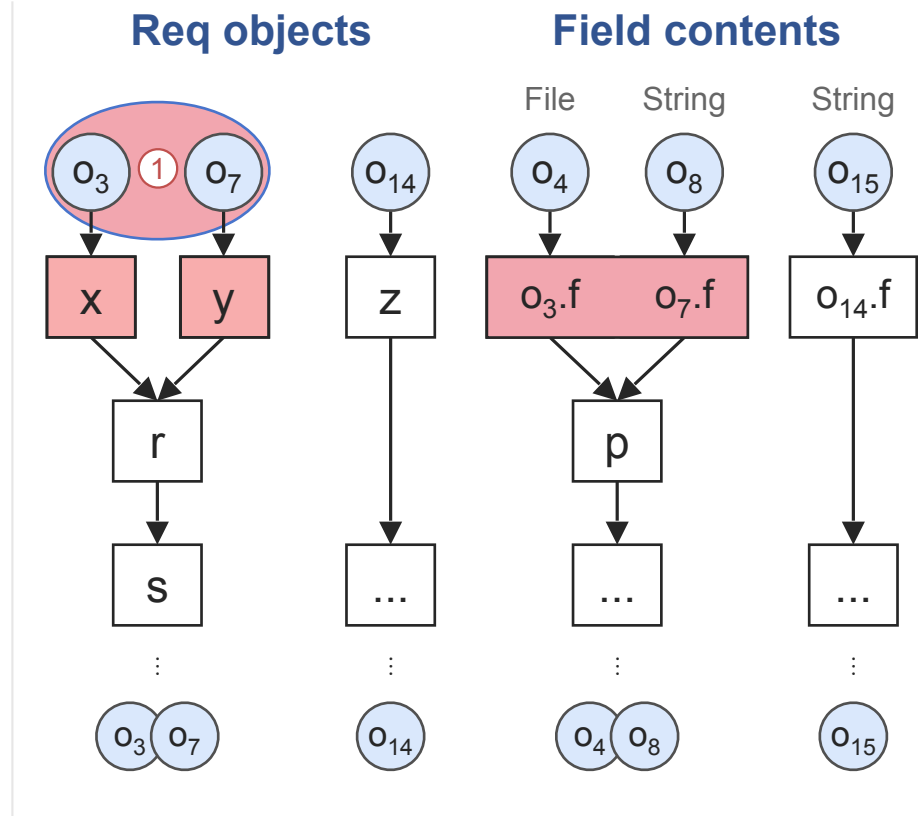
① x points to o_7 , y points to o_3



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

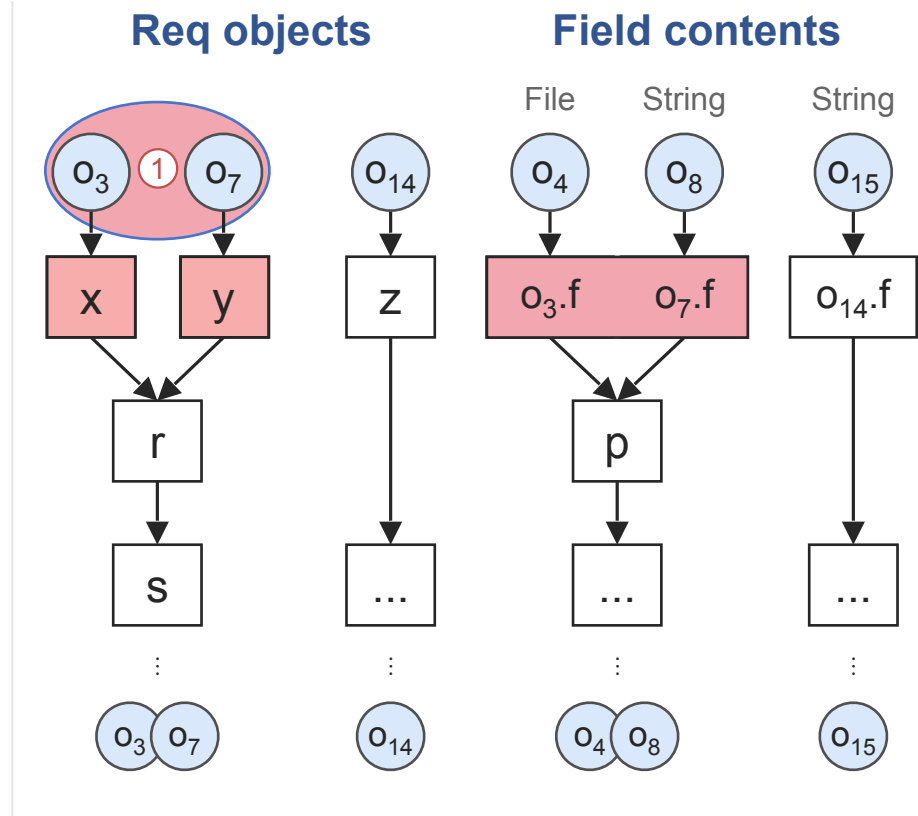
- ① x points to o_7 , y points to o_3
- ② r 's points-to set remains unchanged



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

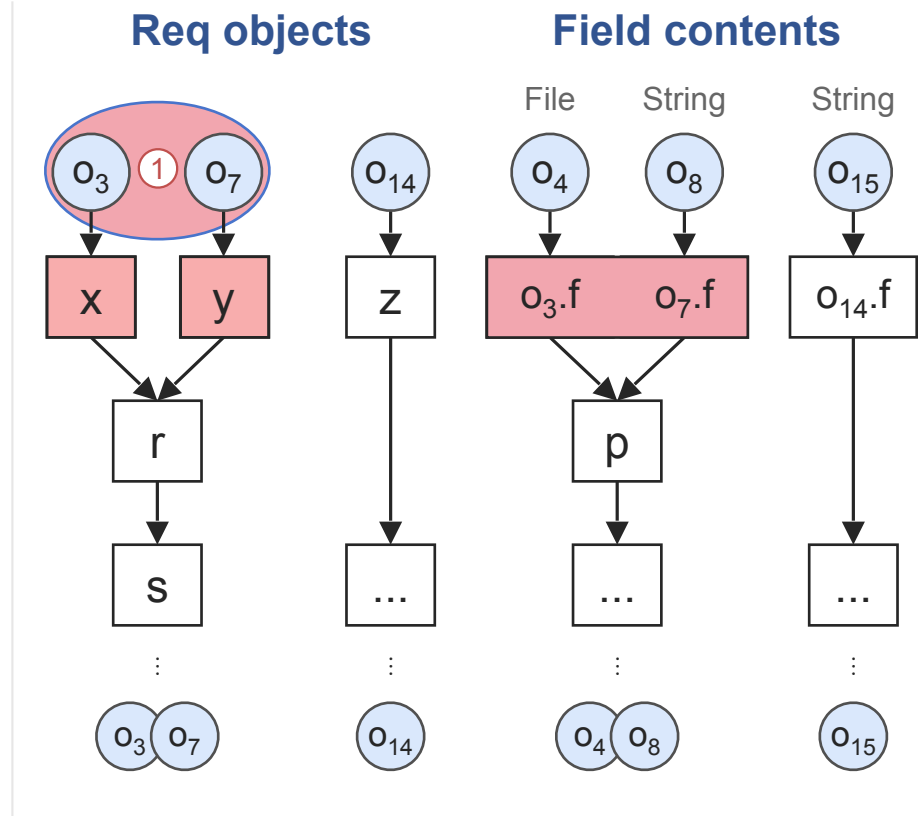
- ① x points to o_7 , y points to o_3
- ② r 's points-to set remains unchanged
- ③ Downstream points-to sets remain unchanged



Valve: Early-Confluent Merging

What happens if we merge o_3 and o_7 ?

- ① x points to o_7 , y points to o_3
- ② r 's points-to set remains unchanged
- ③ Downstream points-to sets remain unchanged



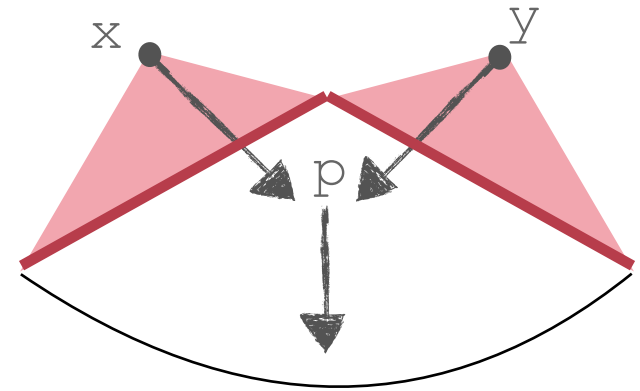
In this example, only x and y lose precision.

Some Notations

Some Notations

- **Confluent Variables**

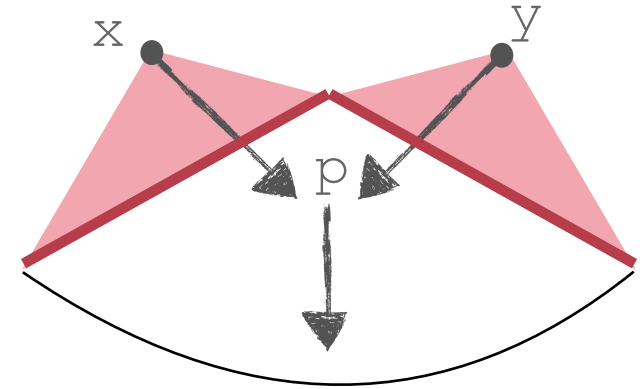
x and y are confluent, if
pointer-flow paths from x and y meet at
some pointer p



Some Notations

- **Confluent Variables**

x and y are confluent, if
pointer-flow paths from x and y meet at
some pointer p



- **Confluence Regions**

Pre-confluence region

Post-confluence region

Confluence frontier



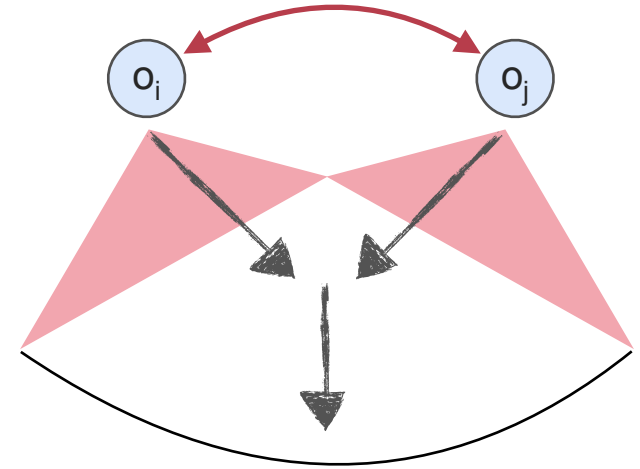
How Merging Introduces Imprecision

- When objects are merged



How Merging Introduces Imprecision

- When objects are merged
Direct imprecision propagates
from merged objects
in the pre-confluence region

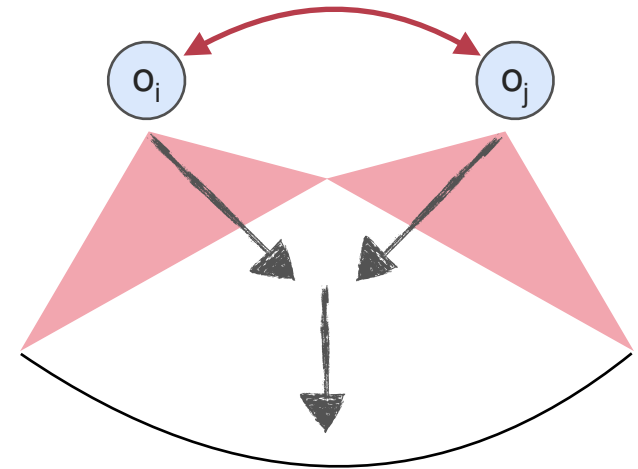


How Merging Introduces Imprecision

- When objects are merged

Direct imprecision propagates
from merged objects
in the pre-confluence region

Indirect imprecision propagates
from field accesses of merged objects
in the pre-confluence region

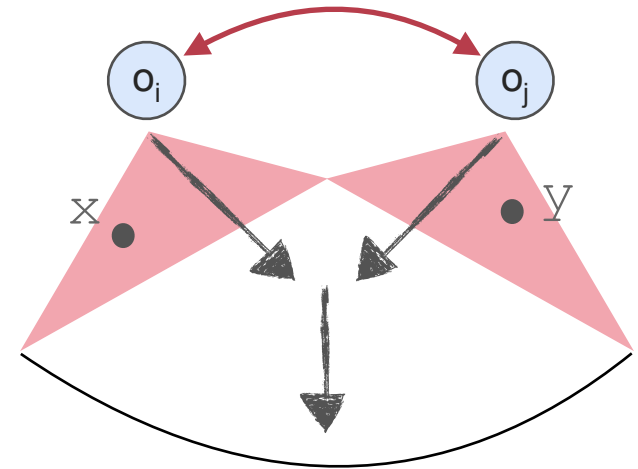


How Merging Introduces Imprecision

- When objects are merged

Direct imprecision propagates
from merged objects
in the pre-confluence region

Indirect imprecision propagates
from field accesses of merged objects
in the pre-confluence region

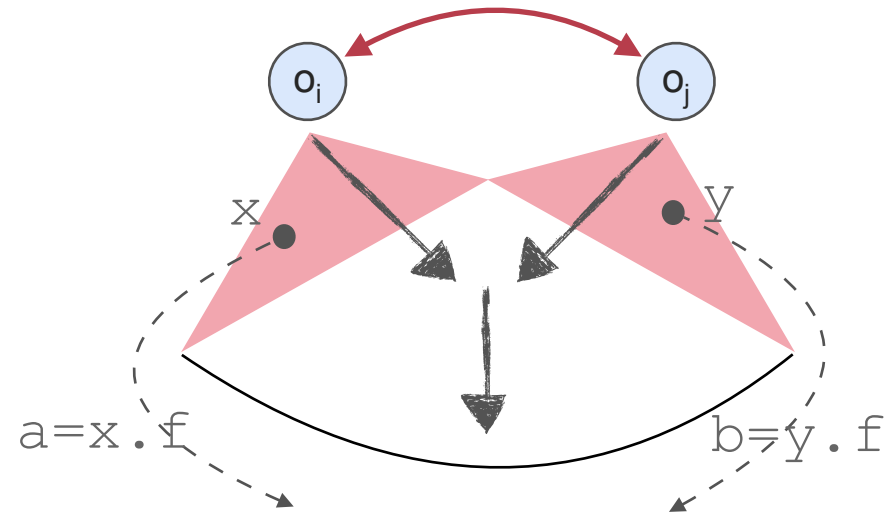


How Merging Introduces Imprecision

- When objects are merged

Direct imprecision propagates
from merged objects
in the pre-confluence region

Indirect imprecision propagates
from field accesses of merged objects
in the pre-confluence region

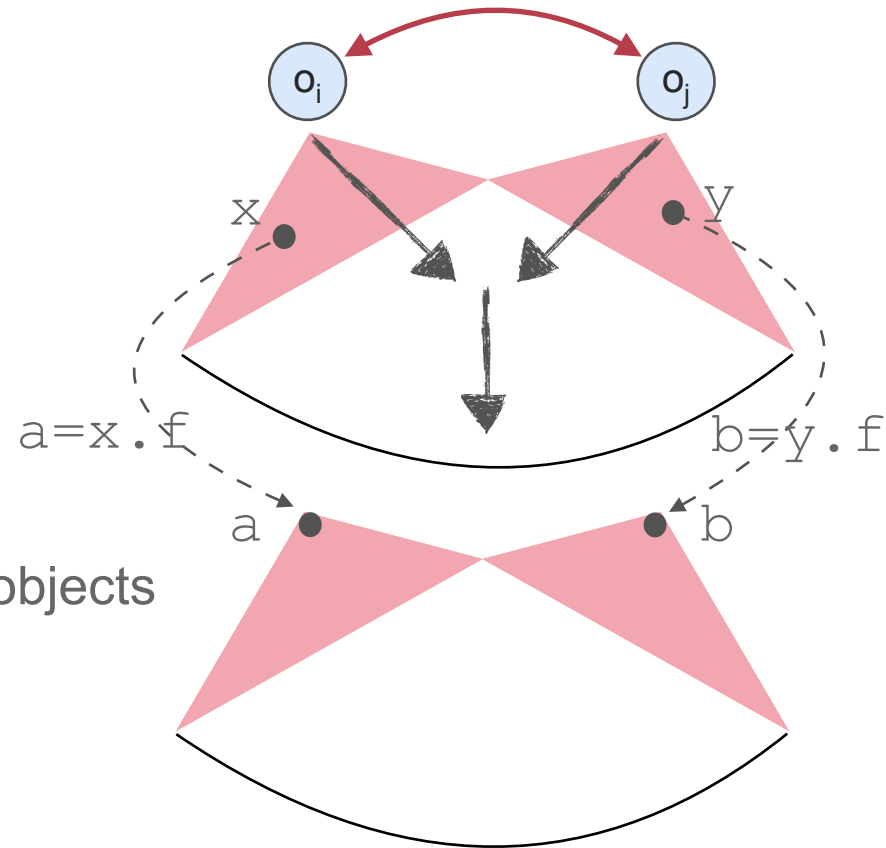


How Merging Introduces Imprecision

- When objects are merged

Direct imprecision propagates
from merged objects
in the pre-confluence region

Indirect imprecision propagates
from field accesses of merged objects
in the pre-confluence region

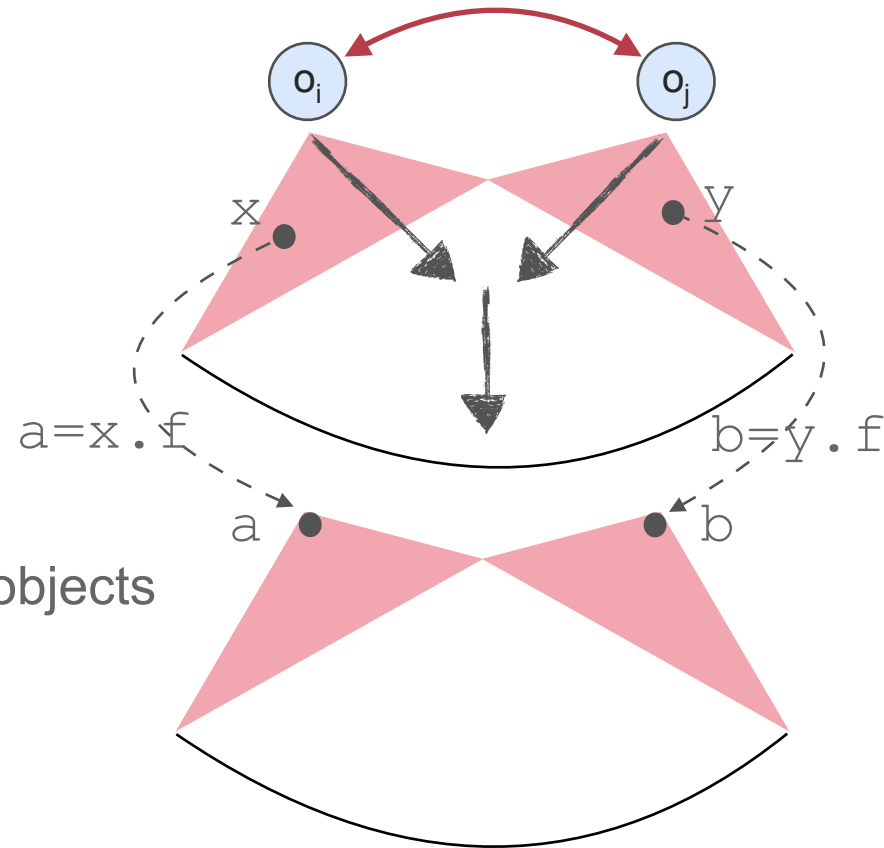


How Merging Introduces Imprecision

- When objects are merged

Direct imprecision propagates
from merged objects
in the pre-confluence region

Indirect imprecision propagates
from field accesses of merged objects
in the pre-confluence region



We aim to confine both types of imprecision.

Early-Confluent Objects

When are objects mergeable

- o_i and o_j are early-confluent if



Early-Confluent Objects

When are objects mergeable

- o_i and o_j are early-confluent if

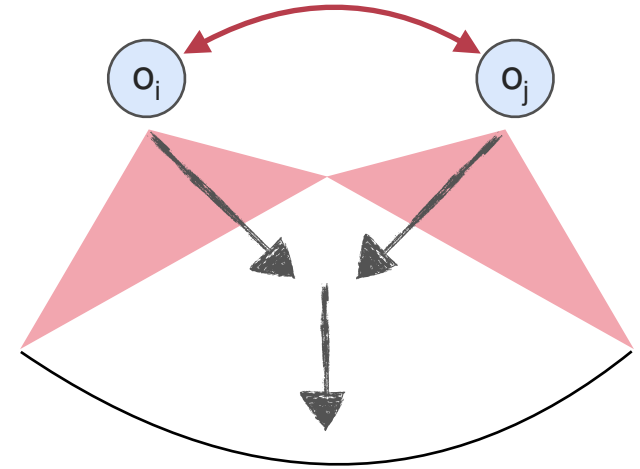
The variables assigned at their allocation-sites, a_i and a_j , are early-confluent



Early-Confluent Objects

When are objects mergeable

- o_i and o_j are early-confluent if
The variables assigned at their
allocation-sites, a_i and a_j , are early-
confluent



Early-Confluent Objects

When are objects mergeable

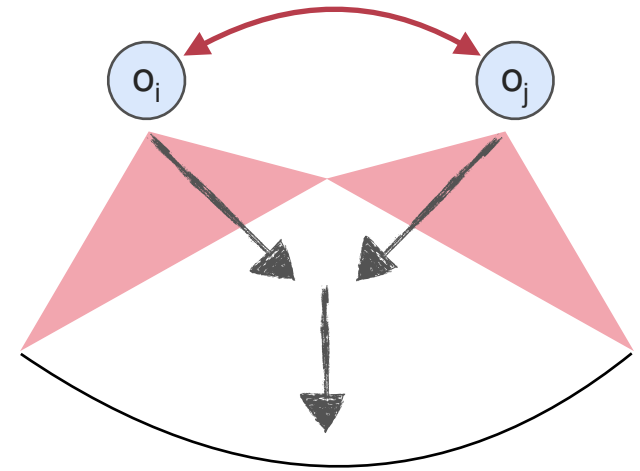
- o_i and o_j are early-confluent if

The variables assigned at their allocation-sites, a_i and a_j , are early-confluent

For every sequence of fields/method calls,

$\bar{f} = f_1 . f_2 \cdot \dots \cdot f_n$, variables loading

$a_i . \bar{f}$ and $a_j . \bar{f}$ are early-confluent



Early-Confluent Objects

When are objects mergeable

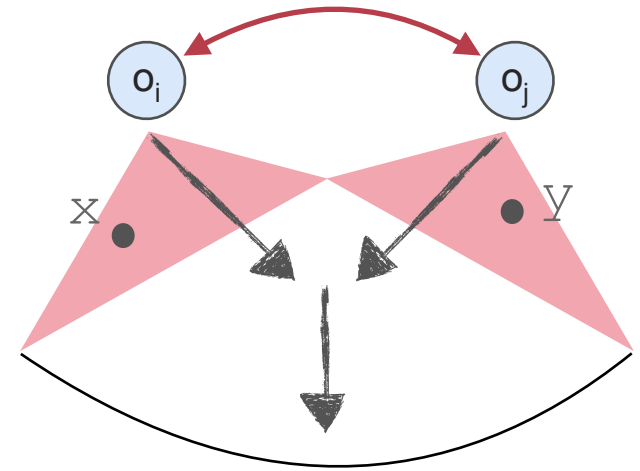
- o_i and o_j are early-confluent if

The variables assigned at their allocation-sites, a_i and a_j , are early-confluent

For every sequence of fields/method calls,

$\bar{f} = f_1 . f_2 \cdot \dots \cdot f_n$, variables loading

$a_i . \bar{f}$ and $a_j . \bar{f}$ are early-confluent

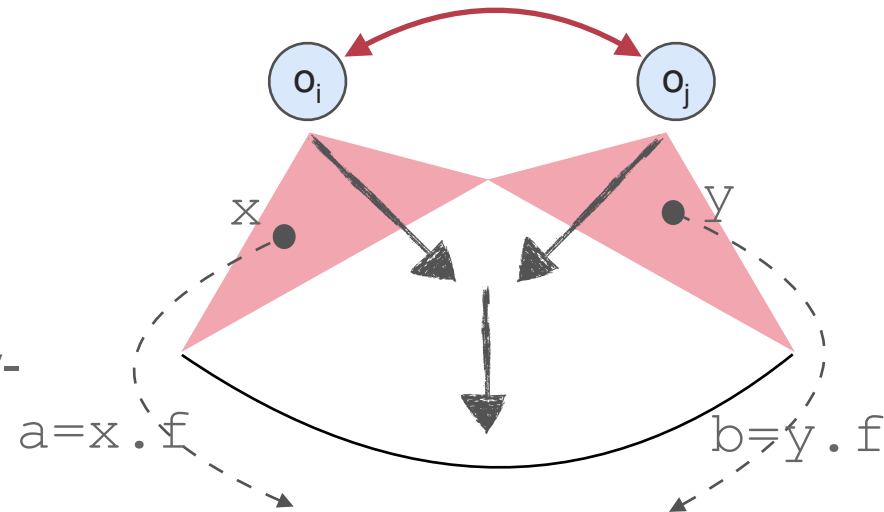


Early-Confluent Objects

When are objects mergeable

- o_i and o_j are early-confluent if

The variables assigned at their allocation-sites, a_i and a_j , are early-confluent



For every sequence of fields/method calls,

$\bar{f} = f_1 . f_2 \cdot \dots \cdot f_n$, variables loading

$a_i . \bar{f}$ and $a_j . \bar{f}$ are early-confluent

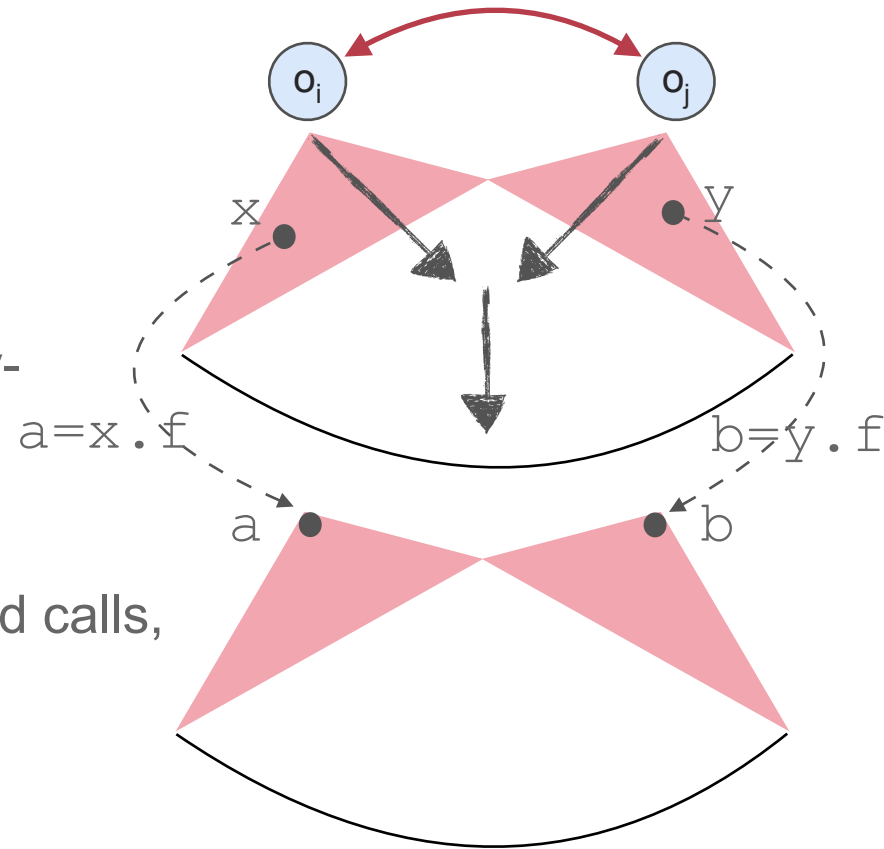
Early-Confluent Objects

When are objects mergeable

- o_i and o_j are early-confluent if

The variables assigned at their allocation-sites, a_i and a_j , are early-confluent

For every sequence of fields/method calls, $\bar{f} = f_1 . f_2 \cdot \dots \cdot f_n$, variables loading $a_i . \bar{f}$ and $a_j . \bar{f}$ are early-confluent



Early-Confluent Variables

When are pre-confluence regions small

- Variables x and y are early-confluent if
They have the same escape sites

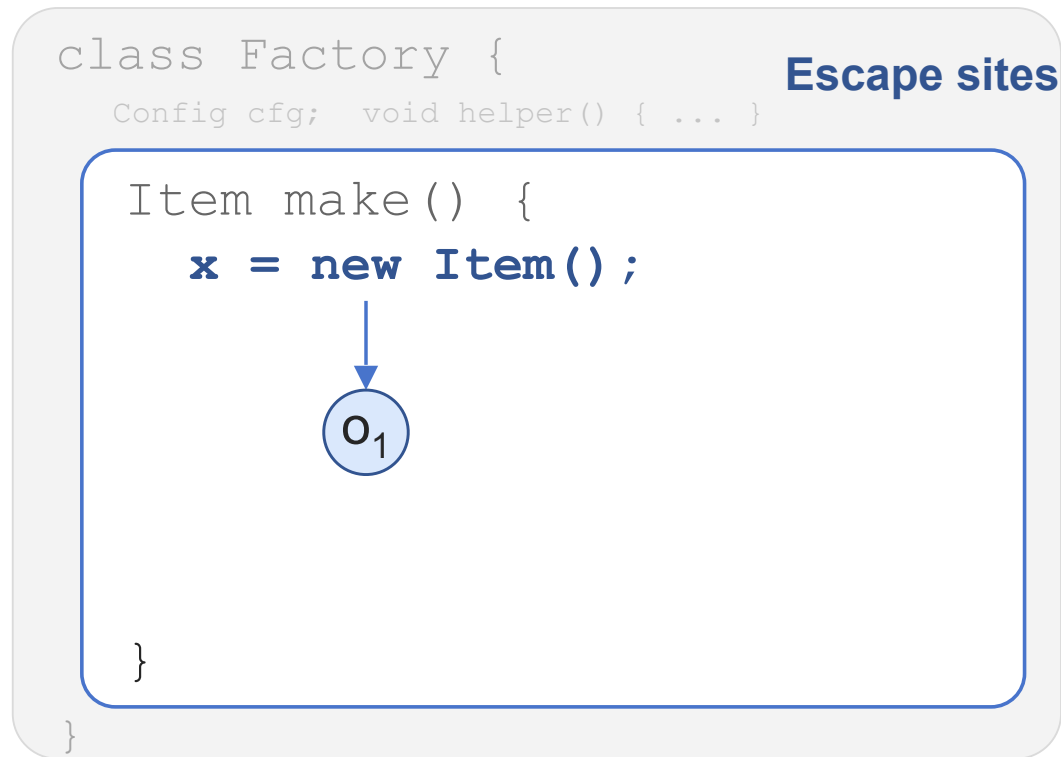
Escape Sites

```
class Factory {  
    Config cfg; void helper() { ... }  
  
    Item make() {  
        x = new Item();  
  
    }  
}
```

Escape sites

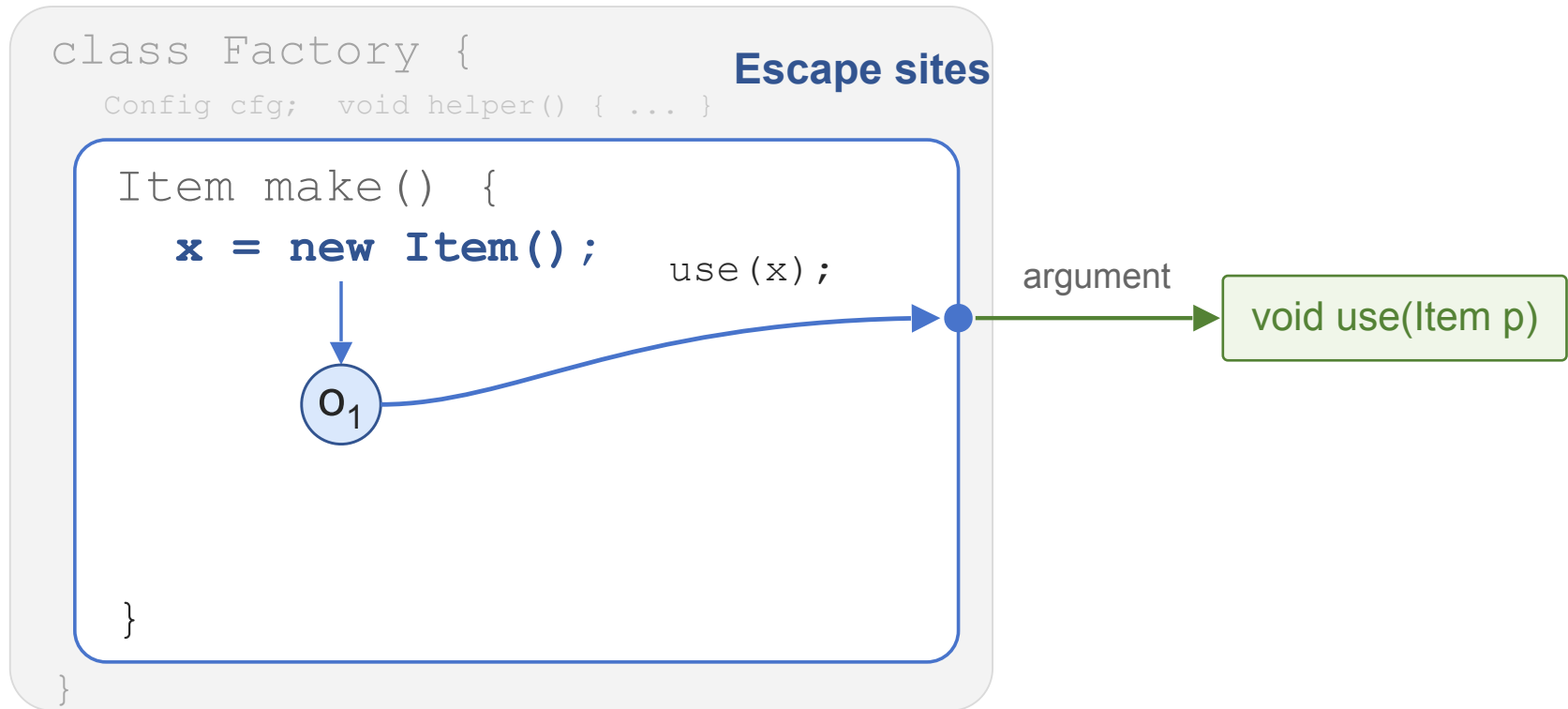
- In any of the 3 cases, o_1 escapes method `make`

Escape Sites



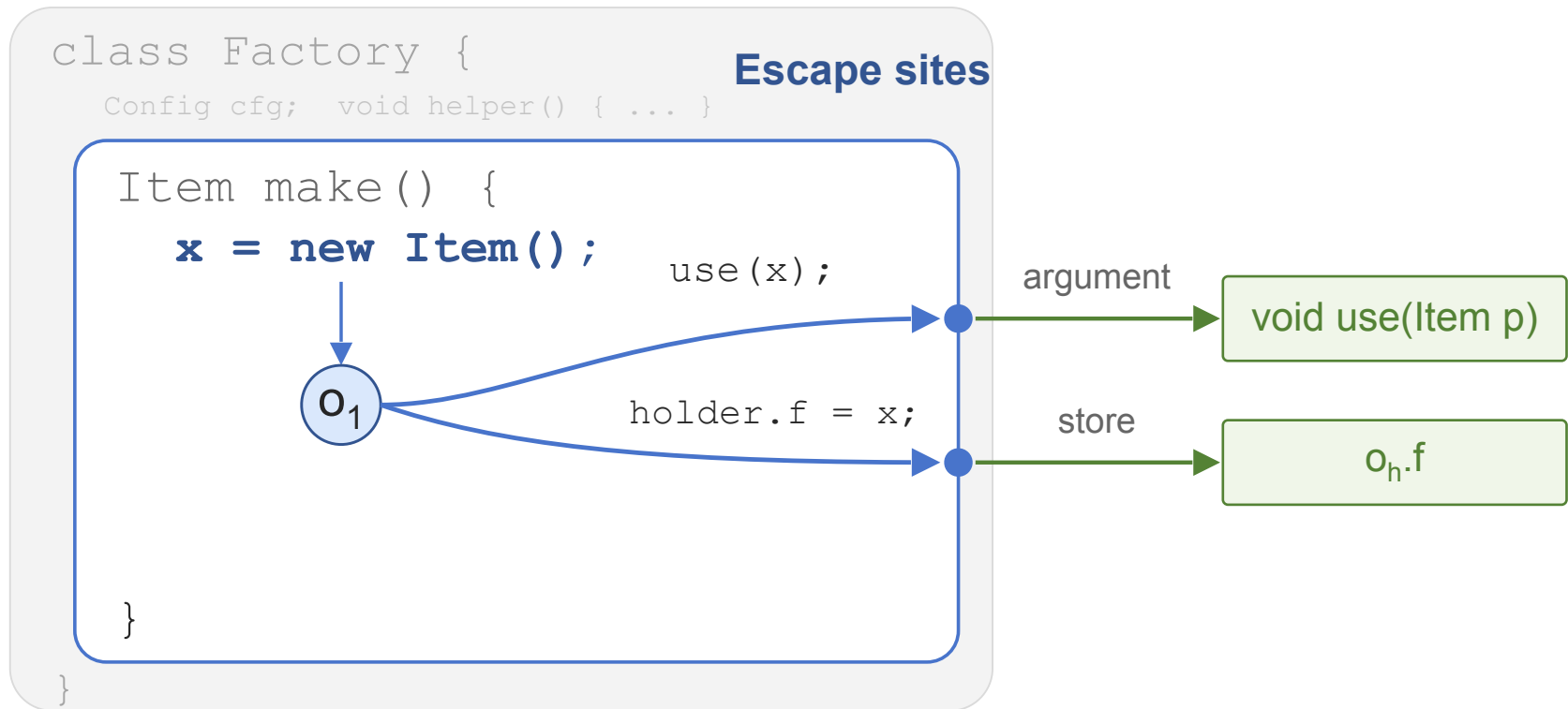
- In any of the 3 cases, o_1 escapes method `make`

Escape Sites



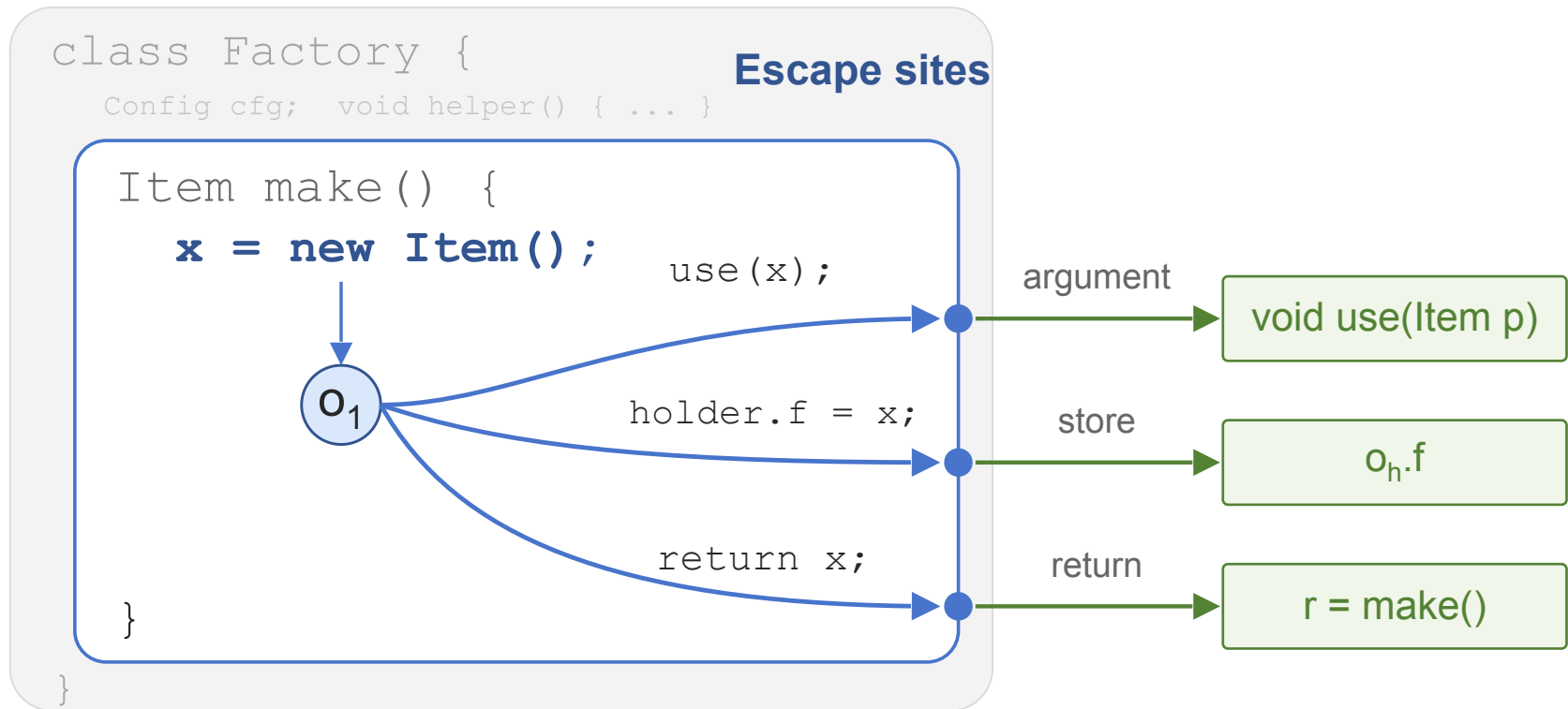
- In any of the 3 cases, o_1 escapes method `make`

Escape Sites



- In any of the 3 cases, o_1 escapes method `make`

Escape Sites

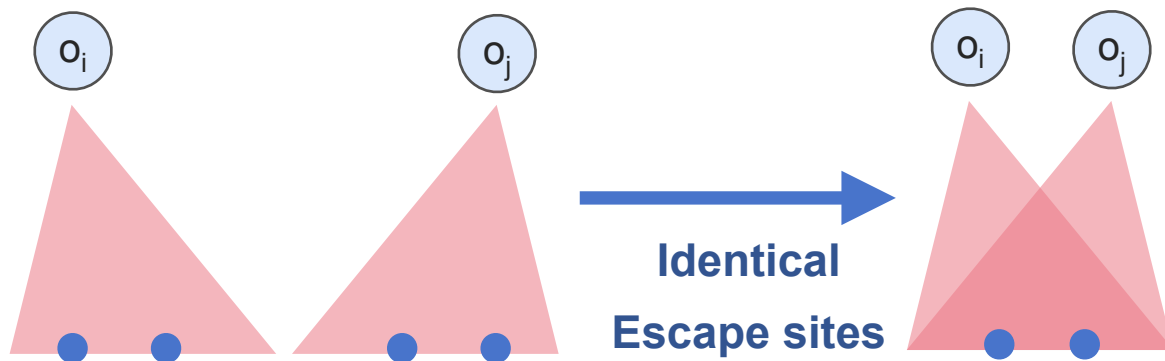


- In any of the 3 cases, o_1 escapes method `make`

What Does “Early” Imply?

Intuitively, if two variables are early-confluent
or, equivalently, have identical escape sites

They converge before escaping the containing method

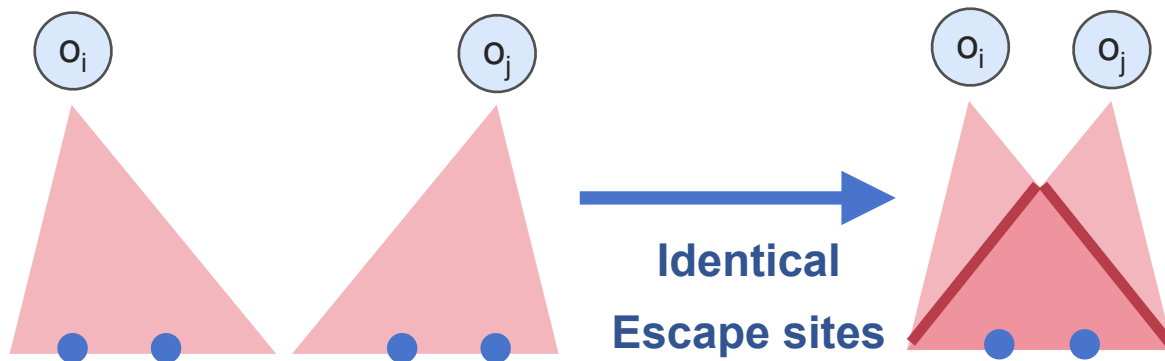


Early-confluent variables $\Rightarrow$ confined pre-confluence region

What Does “Early” Imply?

Intuitively, if two variables are early-confluent
or, equivalently, have identical escape sites

They converge before escaping the containing method



Early-confluent variables $\Rightarrow$ confined pre-confluence region

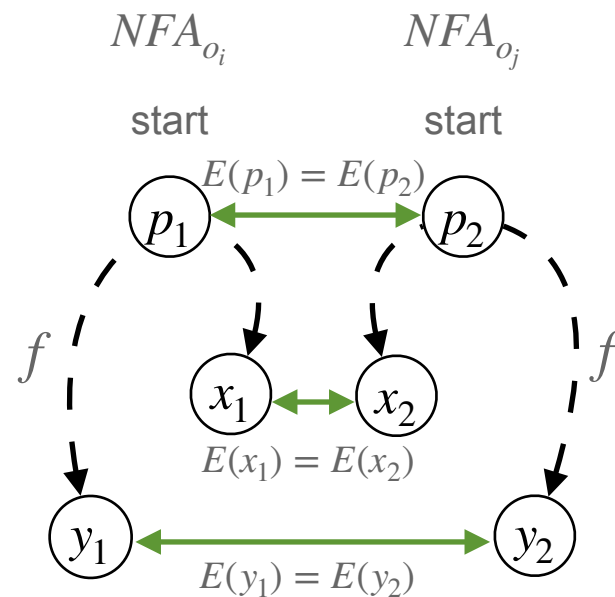
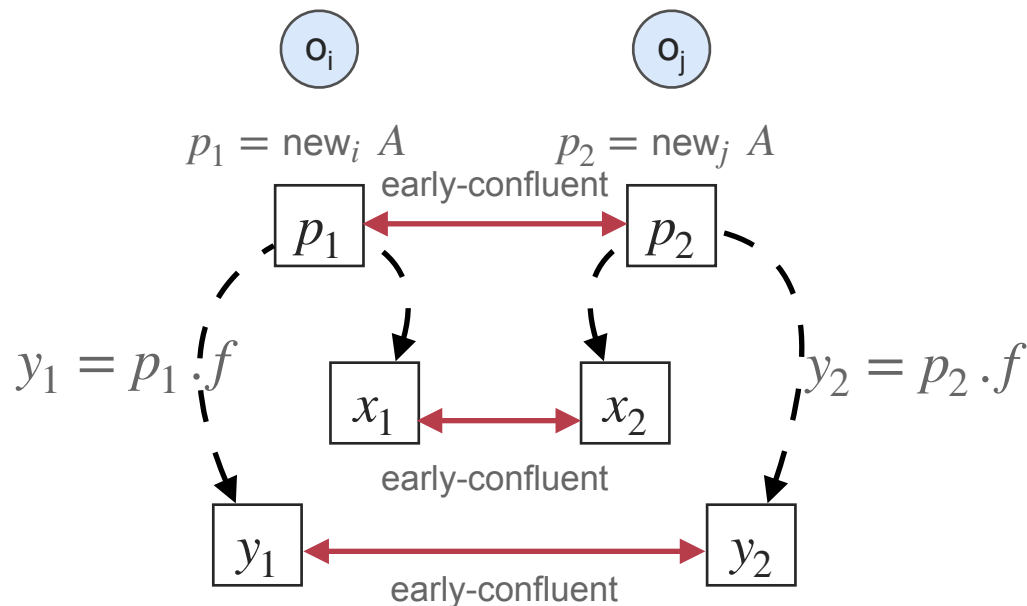
How to Identify Early-Confluent Objects?

- Non-deterministic finite automata (NFA)
equivalence checking

How to Identify Early-Confluent Objects?

Are o_i and o_j
early-confluent?

$NFA_{o_i} \stackrel{?}{=} NFA_{o_j}$



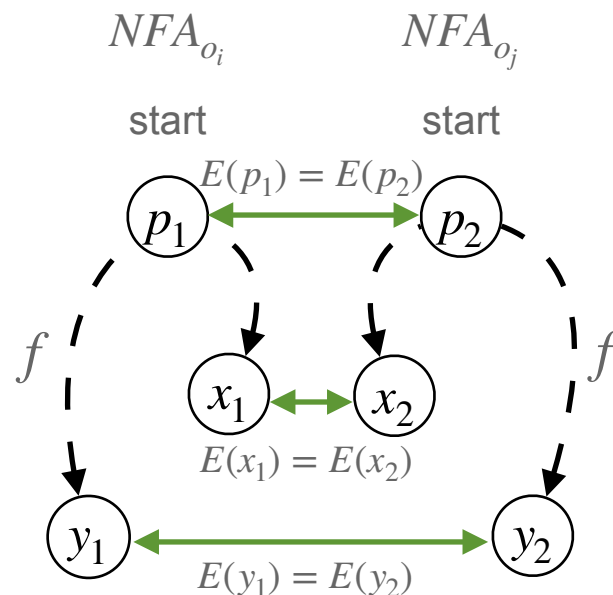
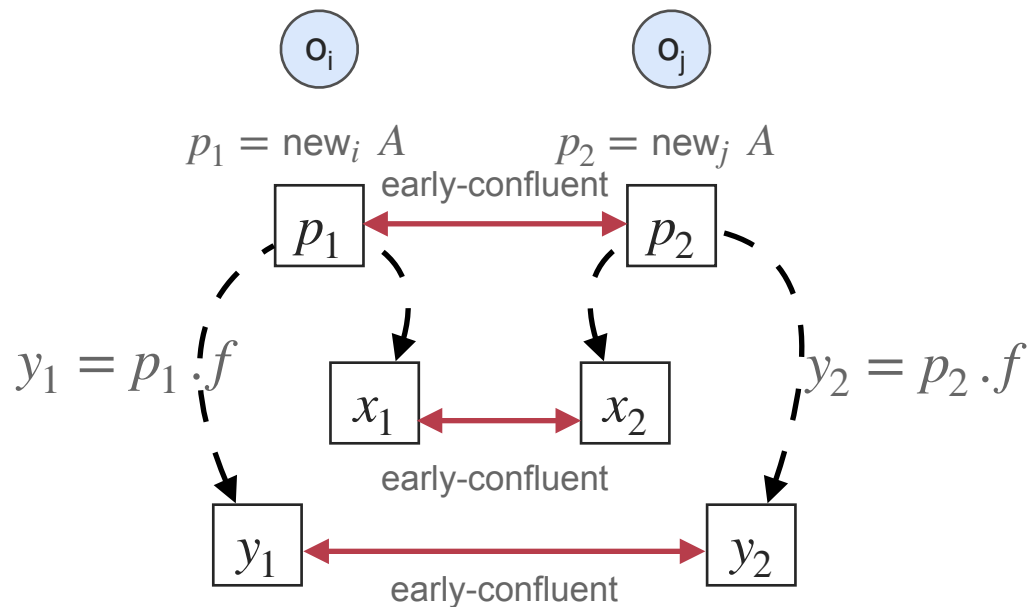
variables

states

How to Identify Early-Confluent Objects?

Are o_i and o_j
early-confluent?

$NFA_{o_i} \stackrel{?}{=} NFA_{o_j}$



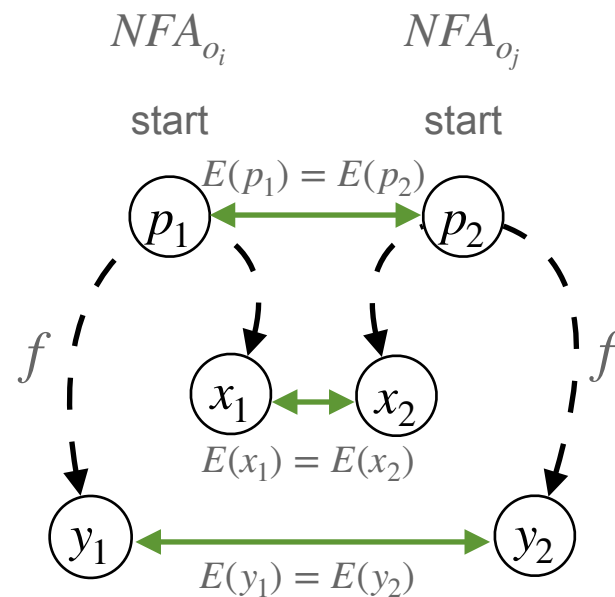
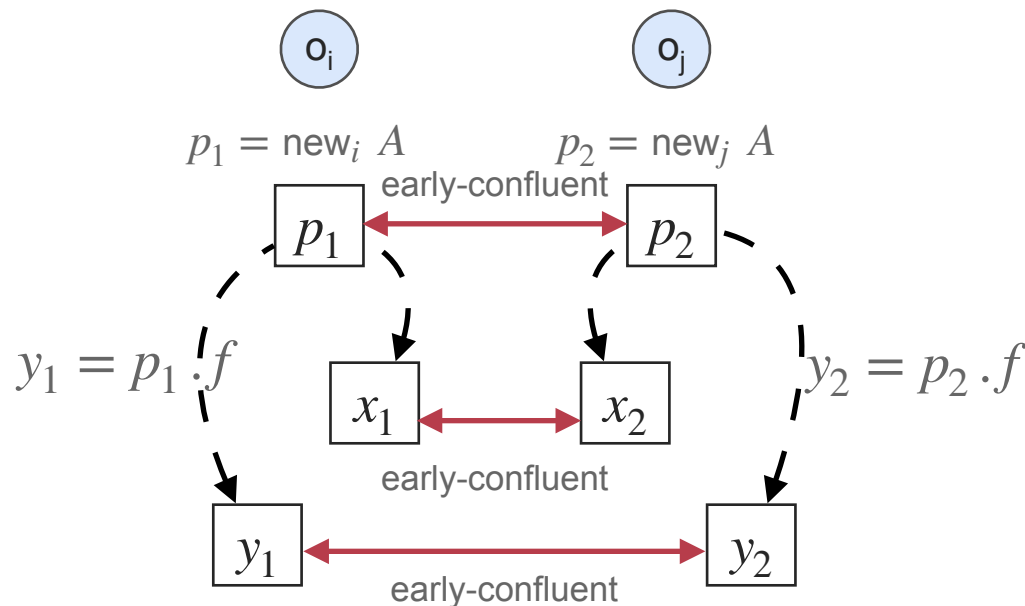
fields/methods

transitions

How to Identify Early-Confluent Objects?

Are o_i and o_j
early-confluent?

$NFA_{o_i} \stackrel{?}{=} NFA_{o_j}$



escape sites

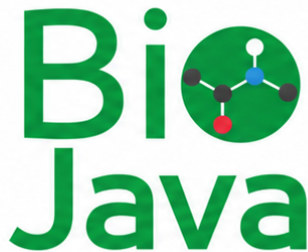
state labels

Evaluation

17 Large Projects



Flink



Renaissance Suite



Settings

- Compared with 2 Heap Abstractions
 - Allocation-site-based heap
 - Mahjong [PLDI'23]
- Combined with 3 Analyses
 - Zipper^e [TOPLAS'20] (Efficiency oriented)
 - Moon [OOPSLA'25] (Precision oriented)
 - Cut-Shortcut [PLDI'23] (Ultra efficient)

4 Analysis Clients

Type-dependent

- Cast resolution
- Call graph

Which *types of objects*
may a variable point-to?

Non type-dependent

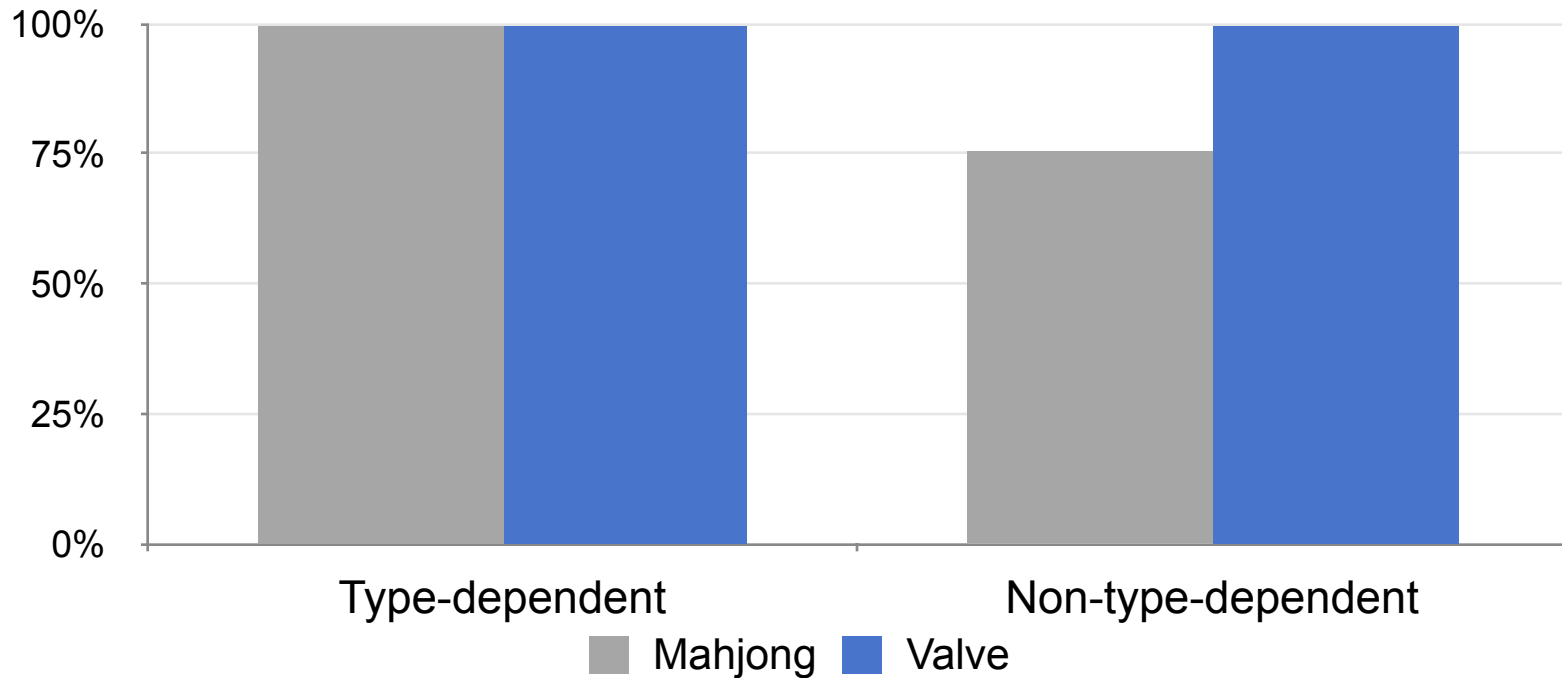
- Thread escape
- Side effects

Which *objects*
may a variable point to?

Smaller client results indicate fewer false positives.

Precision

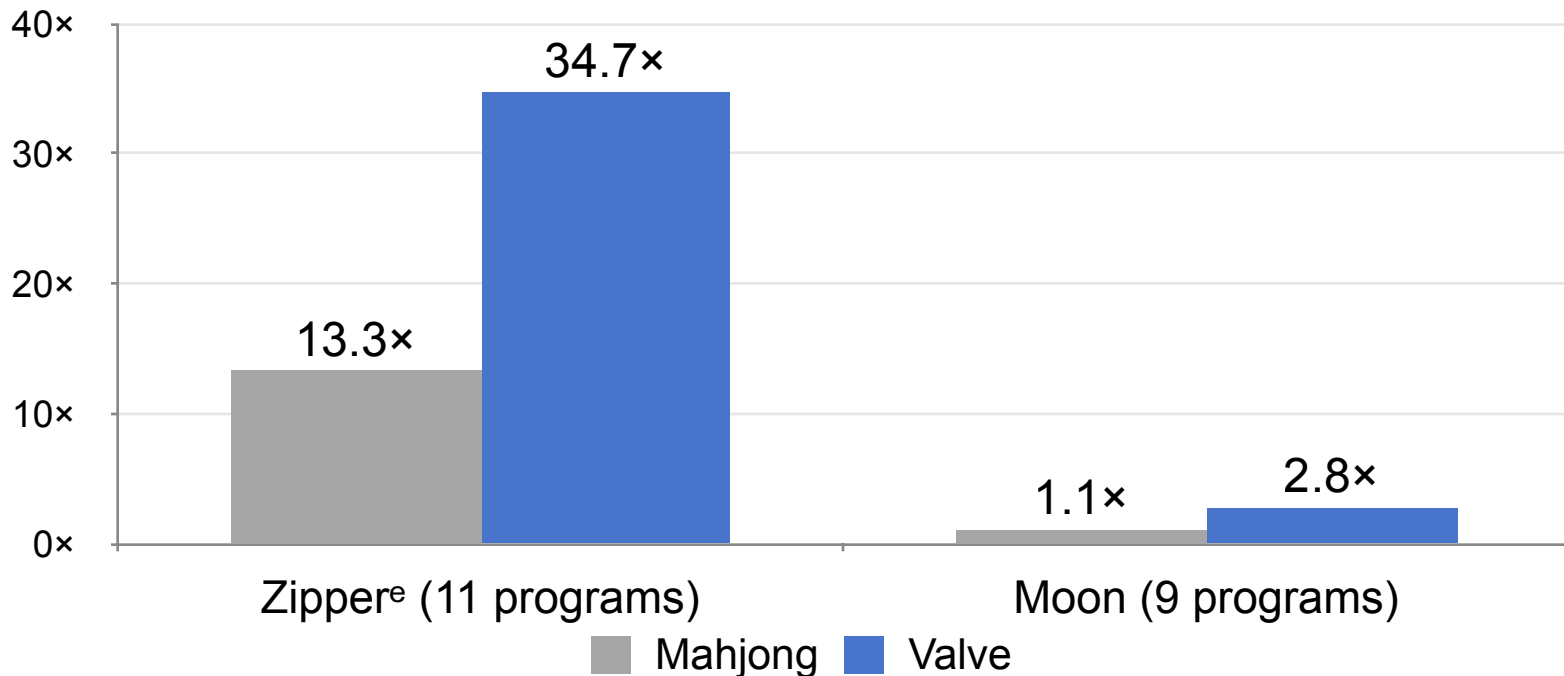
Relative preserved precision across the evaluated analysis clients



High precision is preserved beyond type-dependent clients.

Evaluation: End-to-End Efficiency

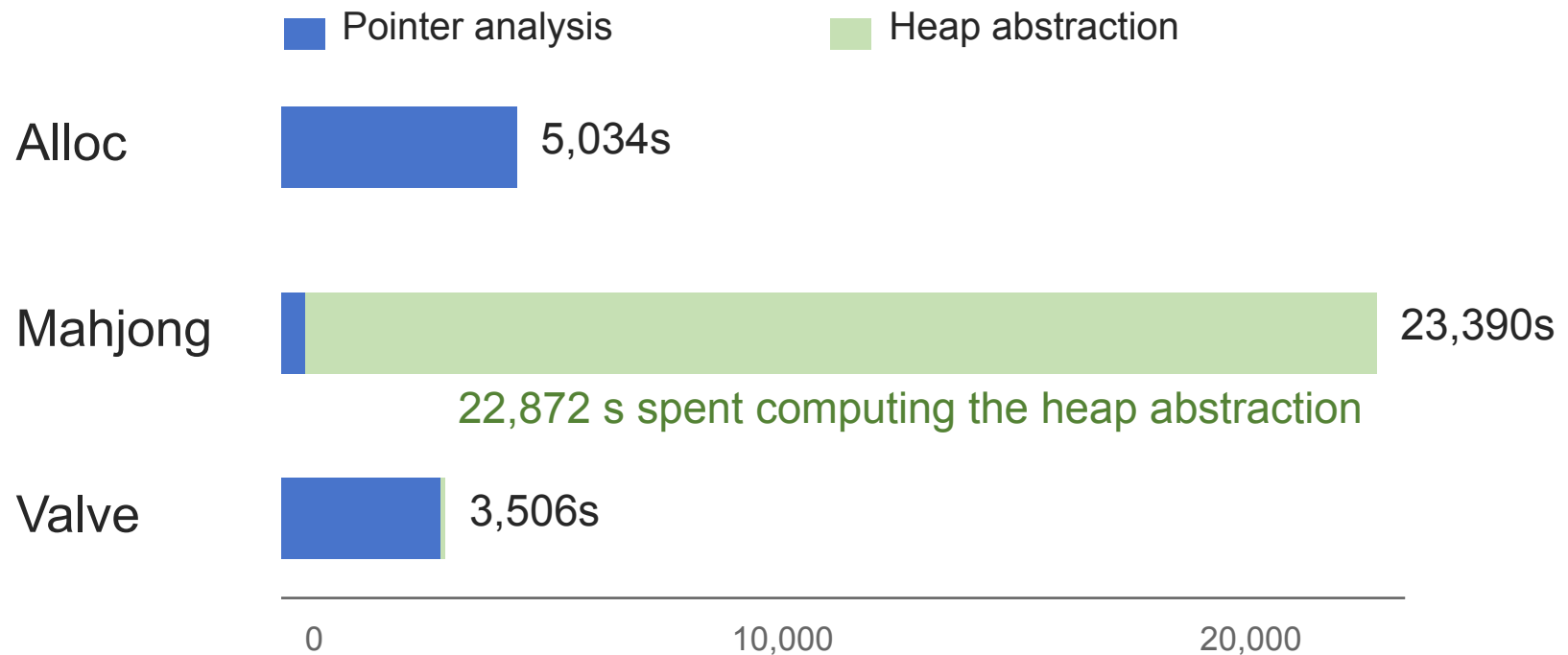
Average speedup over allocation-site-based heap



Substantial speedups, with merging overhead included.

With Cut-Shortcut

flink: total analysis time (seconds; lower is better)



Four largest programs: Valve saves 2,893s on average.

Which Objects Are Merged?

Average reduction in #objects



- Fewer merged objects can still lead to more speedup

Choice of which objects are merged impacts efficiency by affecting the number of objects, contexts, and instance fields.

Conclusion



VALVE

- Merge early-confluent objects
- Identify them using NFA equivalence checking
- High precision beyond type-dependent clients
- End-to-end gains across analysis techniques